A sztochasztikus programozás Monte Carlo módszereiről
Doktori értekezés

Deák István

2004

Tartalomjegyzék

Bevezetés

A sztochasztikus programozás a véletlen jelenlétében hozott optimális döntéshozatal matematikai elmélete [Pr 95], kissé másképpen fogalmazva valószínűségi változókat tartalmazó feladatok vizsgálata és optimalizálása. Bár már korábban is vizsgáltak véletlent tartalmazó döntéshozatali feladatokat, a sztochasztikus programozás rendszeres kutatásának kezdetét Dantzig 1955-ben megjelent cikke [Dan 55] jelenti. Az előzményekről és a korai modellekről jó áttekintést nyújt Prékopa könyve [Pr 95]. Ebben a témában az első könyveket Vajda [Vaj 72], Kall [Kal 76] és Ermoliev [Erm 76] írták. Az utolsó évtizedben jónéhány könyvet írtak a sztochasztikus programozásról, ezek, a teljesség igénye nélkül Birge és Louveaux [BL 97], Frauendorfer [Fra 92], Higle és Sen [HS 96], Infanger [Inf 94], Kall és Wallace [KW 94], Klein Haneveld [Kha 95], Kibzun és Kan [KK 96], Mayer [May 98], Pflug [Pfl 96], Prékopa [Pr 95], van der Wlerk [Wle 95] könyvei. A témában a legújabb kutatási eredményeket a sztochasztikus programozásról háromévenként megrendezett konferenciák közül az utolsó két, válogatott előadásokat tartalmazó kötete tartalmazza (lásd a [WZ 99] és a [De 03b] cikkeket tartalmazó köteteket), valamint a Ruszczynski és Shapiro által szerkesztett összefoglalás [RSh 03].

A sztochasztikus programozásban nemcsak egy használható modell kialakítása, hanem ennek numerikus megoldása is nehéz. Az alkalmazott numerikus eljárások igen széleskörűek, a diszkretizálástól a korlátok megadásáig, mindenféle lineáris és nemlineáris optimalizálási algoritmustól a Monte Carlo módszerekig terjednek. A disszertációban néhány algoritmust közlünk, amelyek a sztochasztikus programozás egyes feladatainak megoldása során felmerülő nehézségek megoldására alkalmasak. Többségük Monte Carlo módszer – a véletlent használják munkaeszközként.

A megoldandó numerikus feladatokat a STABIL valószínűségi korlátot használó modell példáján szemléltetjük. Ezt a magyar közgazdaság elektromos energiai szektor egy problémájának megoldására javasolta Prékopa [PGDP 76] és ez volt az első modell, amely a feladatban szereplő valószínűségi változók korreláltságát is megengedte. A szerző végezte ennek a feladatnak a numerikus megoldását, amelynek során a következő numerikus problémákat merültek fel:

A A többdimenziós normális eloszlás eloszlásfüggvényének kiszámítása,

B a többdimenziós normális eloszlás eloszlásfüggvénye gradiensének meghatározása,

C egy egyenes és egy zajos értékű eloszlásfüggvény által adott felület metszéspontjának megadása,

D egy hatékony nemlineáris optimalizálási algoritmus kifejlesztése a sztochasztikus programozás fő feladatainak megoldására,
E később szükségessé vált megvizsgálni, hogy a párhuzamos számítógépek hogyan használhatók hatékonyan a sztochasztikus programozás feladatainak megoldásában.

Természetesen ezen feladatok általánosabb megfogalmazása is érdekes feladatra vezet, például valamilyen többdimenziós eloszlás esetén egy halmaz valószínűségének, vagy ezen eloszlás gradiensének meghatározása. A STABIL modell egy példájának numerikus megoldása óta eltelt több mint harminc év alatt a szerző a felsorolt numerikus problémákkal foglalkozott, a disszertációban az ezen a feladatok megoldására közöl lehetséges eljárásokat (bár az eljárások a numerikus számítások más területein is felhasználhatók).

A disszertáció első fejezetében a Monte Carlo módszerek alapeszközeivel foglalkozunk, véletlenszámgenerátorokat írunk le. Az **E** alatti feladatra a második fejezet ad egy lehetséges eljárást. Az **A** feladatra vonatkoznak a 3. fejezet eredményei, a **B** és a **C** kérdéskörhöz kapcsolódik a 4. és 5. fejezet anyaga, míg a 6. fejezet a **D** feladatra vonatkozik. Az alábbiakban röviden ismertetjük az egyes fejezetekben leírt eredményeinket.

Véletlenszámgenerátorok

Az első fejezetben két véletlenszám generálási eljárást közlünk a [De 86a] és a [De 90b] cikkek alapján. Az egyik eljárás a folytonos eloszlású véletlen számok generálására használt takarékos módszert [De 81] általánosítja diszkrét valószínűségi változók esetére. Ennek a módszernek az a jelentősége, hogy elég nagy memória felhasználása esetén, tetszőleges diszkrét eloszlású valószínűségi változó egy realizációjának előállítására csak egyetlen darab $[0, 1)$-ben egyenletes eloszlású valószínűségi változóra van szükség.

A második részben kétféle, MIMD típusú párhuzamos számítógépen felhasználható, egyenletes eloszlású véletlen számokat előállító generátort írunk le. Az egyik, Connection típusú számítógépen a Tausworthe [Tau 62] által közölt eltolásos generátor alkalmazható, míg a T-series típusú számítógépeken a Lewis és Payne által közölt [LP 73] általánosított eltolásos generátort használhatjuk. Megmutatjuk, hogy az itt közölt módszerek könnyen implementálhatók számítógépekre, a generátorok jellemzői nehézség nélkül változtathatók, de továbbra is megőrzik nagy periódushosszukat és a generálási eljárás gyorsaságát.

Optimalizálás párhuzamos számítógépeken

A párhuzamos számítógépek fejlődésével kapcsolatban felmerült a kérdés: hogyan lehet egy szekvenciális optimalizálási eljárást hatékonyan használni egy párhuzamos számítógépen. Egy általános algoritmuscsaládot adunk meg, amely egy lazán kapcsolt MIMD típusú párhuzamos számítógépen használható optimalizálásra [De 99] (lásd még a [De 96] és

[De 97b] cikkeket). A közölt eljárás a Zangwill által adott pont-halmaz leképezések elméletének egy következményén alapszik és konvergens és heurisztikus algoritmusok szinte tetszőleges keverékét megengedi. Az algoritmuscsalád előnye az, hogy tetszőleges, iterációkat használó optimalizálási algoritmusra alkalmazható, nincsen Descartes-szorzat jellegű kikötés sem az algoritmusra, sem a megoldások halmazára (mint a Bertsekas és Tsitsiklis által javasolt eljárásban [BT 89]), a rendelkezésünkre álló számítógép processzor számától függetlenül használható és teljesen aszinkron üzemmódban működik (nem kell várnia más processzorok eredményeire). Az eljárás kommunikációs igénye kicsi, így ideálisan alkalmazható lazán kapcsolt MIMD gépekre, például munkaállomásokból kialakított ideiglenes hálózatokra is.

Az eljárás alkalmazható olyan sztochasztikus programozási feladatok numerikus megoldására, amelyekben egy (vagy néhány) nehezen kiszámítható (zajos) függvény is szerepel. A rendelkezésünkre álló processzorok közül néhány zajos függvényértékeket határozhat meg, más processzorok a zajos függvényértékek segítségével regressziós becsléseket határozhatnak meg a nehéz függvények közelítésére, míg a maradék processzorok foglalkozhatnak a közelítő feladatok optimalizálásával.

Halmazok valószínűségének kiszámítása

A statisztikában, megbízhatósági feladatokban, sztochasztikus programozásban gyakran van szükség többdimenziós térben elhelyezkedő halmazok valószínűségének meghatározására. Kutatásainkban csak az alkalmazásokban fontos szerepet játszó többdimenziós normális eloszlással és Monte Carlo módszerekkel foglalkoztunk (mivel ezek alkalmazhatók magasabb dimenzióban is). Monte Carlo módszereknek halmazok valószínűsége kiszámítására való alkalmazása területén több megközelítés ismert, a szerző [De 80a] ortonormált becslései, Szántai [Sz 86] szita formulán alapuló eljárása, Gassmann [Ga 88] ezeket együtt használó és általánosító eljárása, Prékopa egyenlőtlenségeket alkalmazó megközelítése [Pr 88a], [Pr 95], Bukszár és Prékopa eredményei [BP 01]. Ezeknek és más ismert eljárásoknak a numerikus vizsgálata és összehasonlítása található a [GDS 02] cikkben. Hasonló összehasonlításokat tartalmaznak a [HFR 96], [Vij 97] cikkek. Meg kell említeni Lovász és Simonovits kutatásait is, amelyekben polinomiális algoritmust keresnek halmazok valószínűségének meghatározására, a legújabb eredmények szerint már csak $n^{4.5}$ nagyságrendű műveletre van szükség [LS 93], [KLS 97], [Lov 99].

Az itt közölt algoritmusok az ortonormált becsléseknek tetszőleges poliéderre, ellipszoidra és körkúpra történt általánosítását tartalmazza a [De 00] cikk alapján (további numerikus eredményeket, illetőleg a szubrutinrendszer leírását tartalmazzák a [De 98c], [De 03a] cikkek). A számítógépes futások eredményeit összegezve mondhatjuk, hogy a közölt eljárások 20 dimenziós halmazok esetében nagyon hatékonyak (négy tizedesre pontos eredmények kaphatók 1 másodperc alatt), és gyakorlatilag használható eredményeket lehet kiszámítani 100 dimenzióig. A megbízhatósági számításokban használt nagyon kis, 10^{-4} körüli valószínűségekre is gyorsan lehet, legalább egy értékes jegyet tartalmazó

becslést kapni. A kifejlesztett számítógépes NORSET szubrutinrendszert beépítették a University of Zurich operációkutatási intézetében kifejlesztett SLP-IOR [KM 96] sztochasztikus programcsomagba.

Egydimenziós közelítések

A **C** alatti kérdéskör megoldására jónéhány megoldó módszert javasoltak már, mint például a sztochasztikus approximáció eljárását [RM 51], [Dvo 56], [KC 78], de természetesen használható akármilyen egyenlet megoldási eljárás, amelyet zajos függvények esetére megfelelően módosítottak (lásd például [AF 01]).

Egy egyenes és egy zajos felület metszéspontjának meghatározására használhatunk regressziót is, pontosabban L_2 minimális normájú lineáris vagy kvadratikus alakú közelítéseket [De 89], ezeknek a becsléseknek különböző formáit pedig a [De 01b] cikkben írtuk le. A becslések meghatározásával együtt nemcsak a kvantilis meghatározására, hanem az iránymenti deriváltak, illetőleg a gradiens kiszámításának feladatára is eljárást kapunk [De 98b], [De 98a]. Az ilyen becslések használatával a párhuzamos optimalizálás esetén a számítások felgyorsítását is elérhetjük [De 96].

A fejezetben a [De 97a] cikk alapján leírjuk, hogyan lehet könnyen kiszámítható, viszonylag kis hibájú közelítéseket konstruálni az egydimenziós normális eloszlás eloszlásfüggvényére a legkisebb négyzetek módszere segítségével, majd a [De 98b] cikk alapján regressziós becsléseket adunk a többdimenziós normális eloszlás eloszlásfüggvényének egy egyenes mentén felvett értékeire.

Szukcesszív regressziós approximációk egydimenzióban

Egydimenziós egyenletek gyökének meghatározásával foglalkozunk ebben a fejezetben. Egy iteratív eljárást neveztünk el a szukcesszív regressziós approximációk módszerének (röviden SRA algoritmusnak), amely a következő alapvető lépésekből áll. Feltesszük, hogy néhány pontban már kiszámítottuk a függvény értékét, ekkor

i) a pontok és függvényértékek segítségével egy (minimális L_2 normájú) regressziós közelítést határozunk meg,

ii) az eredeti függvényt helyettesítjük a regressziós becsléssel és ezt a közelítő feladatot oldjuk meg,

iii) a közelítő feladat optimális megoldását hozzáadjuk a regresszió kiszámításához használt ponthalmazhoz és újabb regressziót határozunk meg.

Ez a módszer egyre pontosabb közelítéseket ad a gyök közelében. Az SRA lényegében a legkisebb négyzetek módszerét és a nagy számok törvényét kapcsolja össze. A legkisebb négyzetek módszerét Gauss használta először, több mint kétszáz évvel ezelőtt bolygók pályáinak meghatározására, használata ma is széleskörű a numerikus számításokban [DS 66], [Bjo 96], [LH 95]. A legkisebb négyzetek eljárása biztosítja azt, hogy a közelítések könnyen kiszámíthatók.

A nagy számok törvénye lényegében azt mondja ki, hogy különböző feltételek mellett az átlagolás csökkenti a hibát. Természetesen, különböző tudományterületeken az átlagolás szokványos eljárás, hiszen az egész statisztika ezen alapszik. Például az irányításelméletben, előrejelzésben és a paraméterbecslésben nagyon gyakran használják ezt az eljárást, lásd Polyak és Juditsky [PJ 92], Györfi és Walk [GW 96], Spall [Spa 92], Yin [Yin 91], a sztochasztikus approximációban Kushner és Yang [KY 93], [KY 95], általánosabban az adaptív módszerek között [BMP 90], az optimalizálás elméletében Ermoliev [Erm 76] és Ruszczynski [Rus 80] kutatásait. A fő kérdés az, hogy milyen feltételek mellett áll fenn nagy számok törvénye, illetőleg milyen gyors a konvergencia. Bizonyos gyenge összefüggés esetén (L-keverő sorozatokra) Gerencsér [Ger 92] publikált nemrégiben eredményeket.

Az *SRA* eljárást először egydimenziós (pontosan kiszámítható) függvények gyökének meghatározására írjuk le a [De 01a] alapján, majd a zajos függvény értékek esetére. Ez az eljárás természetesen szorosan összefügg az elsőrendű szükséges optimalitási feltételeken keresztül a sztochasztikus programozás feladataival, vagy általában az optimalizálással. Bebizonyítottuk, hogy az *SRA* eljárás konvergál determinisztikus függvényértékek esetén [De 01a]. Az elméleti eredményeket számítógépes eredményekkel is alátámasztottuk [De 98a], [De 98d], [De 01b], de jelenleg még nincsen teljes konvergenciabizonyítás a zajos függvényérték esetére. Mindenesetre az itt elért eredmények alapján tértünk át az n-dimenziós általánosításra.

Szukcesszív regressziós approximációk a sztochasztikus programozásban

Az *SRA* eljárást alkalmaztuk sztochasztikus programozási feladatokra – ez jelenleg egy heurisztikus eljárás, de hatékony eszköznek bizonyult a számítási eredmények alapján. Mindegyik megvizsgált esetre jellemző, hogy egy vagy két nehezen (zajosan) kiszámítható függvényünk volt, amit egy regressziós becsléssel helyettesítettünk, és a közelítő feladat optimális eredményét visszacsatoltuk az alaphalmazba.

Az *SRA* algoritmus segítségével a következő sztochasztikus kvadratikus programozási feladatot lehet megoldani:

$$\begin{aligned} \min E[\mathcal{Q}(\mathbf{x}, \boldsymbol{\xi}_0)] + \mathbf{x}'Q_0\mathbf{x} + \mathbf{c}_0'\mathbf{x} & \\ \text{f.h. } \mathbf{x}'Q_i\mathbf{x} + \mathbf{c}_i'\mathbf{x} &\leq b_i, i \in I_1, \\ P\left\{h_i(\mathbf{x}, \boldsymbol{\xi}_i) \leq 0\right\} &\geq p_i, i \in I_2, \\ \mathbf{x} \geq 0, & \end{aligned} \tag{1}$$

$$\text{ahol } \mathcal{Q}(\mathbf{x}, \boldsymbol{\xi}_0) = \{\min \mathbf{q}'\mathbf{y} \mid T\mathbf{x} + W\mathbf{y} = \boldsymbol{\xi}_0, \mathbf{y} \geq 0\}.$$

Ennek az alapfeladatnak következő változatait oldottuk meg numerikusan az *SRA* algoritmussal: a STABIL modellt, a kétlépcsős modellt és a Prékopa által megfogalmazott vegyes modellt.

A STABIL modell és hasonló, valószínűségi korlátokat tartalmazó feladatok megoldására nagyszámú optimalizálási eljárást javasoltak, illetőleg ezek számítógépes alkalmazását és hatékonyságuk vizsgálatát is elvégezték, lásd például [PGDP 76], Rapcsák [Rap 77], Szántai [Sz 88], Komáromi [Kom 87], Kall és Mayer [KM 96], Gaivoronski [Gai 86]. Ezek az eljárások általában egy nemlineáris optimalizálási módszer olyan változatai, amelyeket a pontatlan (zajos) függvényértékek esetén is használni lehet. Az ismert számítógépes futások alapján pillanatnyilag úgy tűnik, hogy a legjobb megoldó eljárás az általánosított redukált gradiens módszer (GRG), amelyet a University of Zurich operációkutatási intézetében Kall és Mayer által kifejlesztett SLP-IOR sztochasztikus programozási programcsomagban [KM 96], [May 98] valósítottak meg, bár a Szántai által kifejlesztett sztochasztikus programozási rendszer [Sz 88] is elég hatékonynak tűnik. Az SRA algoritmus használata a valószínűségi korlátos feladatoknál valószínűleg nem hatékonyabb az ismert eljárásoknál, de egy új, alternatív megoldási lehetőséget kínál.

Ami a másik, széles körben alkalmazott sztochasztikus programozási feladattípust, a kétlépcsős feladatot illeti, szintén sok megoldási módszer ismert, lásd például Higle és Sen [HS 96], Wets [Wet 83], Kall és Wallace [KW 94], Birge és Louveaux [BL 97], valamint Mayer eredményeit [May 98]. A feladatban szereplő valószínűségi változók eloszlását általában vagy diszkrétnek tekintik, vagy folytonos eloszlás esetén diszkretizálják és az így keletkezett nagyméretű lineáris feladatot oldják meg. Ezen megközelítések közül Ruszczyński eredményeit emeljük ki, amelyben a lineáris programozási feladat méretét korlátozni tudja [RSw 97].

Az általunk javasolt szukcesszív regressziós approximációk módszerében ezzel mintegy ellentétesen járunk el, a szakaszosan lineáris várható pótlás függvényét egy kétszer folytonosan differenciálható függvénnyel közelítjük. A módszert leíró cikkeink ([De 98b], [De 01a], [De 03b], [De 03c]) és számítógépes tapasztalataink alapján úgy tűnik, hogy az SRA használatával hatékonyabb algoritmus is készíthető. Egy közepes méretű feladatnál a közelítő optimális megoldás körülbelül azonos idő alatt (fél perc - három perc) kapható meg, de a függvényérték két tizedessel pontosabb az eddig ismert eredményeknél. Előnye az ajánlott módszernek, hogy korrelált valószínűségi változókat is lehet kezelni és a valószínűségi változó dimenziójától sokkal kevésbé függ az eljárás gyorsasága. Megemlítjük még, hogy a szerző által ismert eddigi legnagyobb kétlépcső feladat (89 első lépcsős változó és 86 dimenziós véletlen vektor, lásd [SDC 93]) méreteihez hasonlítható nagyságú feladatokat is meg tudtunk oldani – egy 100 első lépcsős változót és 120 dimenziós valószínűségi vektort tartalmazó kétlépcsős feladatot a Függelékben közlünk.

Prékopa javasolt egy feladatot a kétlépcsős modell egy hiányosságának kezelésére. A kétlépcsős modellben ugyanis szerepel egy indukált feltételeknek nevezett, indirekt módon megadott követelményrendszer, amely azt biztosítja, hogy a második lépcsős feladatnak van véges optimuma az elsőlépcsős változók és a második lépcsőben szereplő valószínűségi változók minden lehetséges értékére. Prékopa ebben a feladattípusban csak azt követeli meg, hogy elég nagy valószínűséggel legyen megengedett megoldása és véges optimuma a második lépcsős feladatnak. Az ennek alapján felírt feladat tartalmaz valószínűségi korlá-

tot is, és a kétlépcsős feladatra jellemző várható pótlási költséget is, ezért nevezzük vegyes modellnek. A szukcesszív regressziós approximációk módszerével ez a feladat is megoldható (eddig nem volt ismert megoldó algoritmus). Valószínűleg a paraméterbecslésben is felhasználható az eljárás [AM 79].

Az *SRA* algoritmus használata során megfogalmaztuk azt a numerikus eredményekkel alátámasztott sejtésünket, hogy a pontosság a lehető leggyorsabban nő, a végeredmények hibája $O(1/\sqrt{M})$ nagyságú, ahol M az egész eljárás alatt kiszámított zajos függvényértékek száma volt.

A 6. fejezetben közölt eredményeinket a [De 03b], [De 03c] és a [De 04] cikkek alapján írtuk le (további részletek és numerikus eredmények találhatók a [De 98a], [De 02] cikkekben). Az *SRA* algoritmust alkalmaztuk valószínűségi korlátos feladatokra, kétlépcsős feladatokra és Prékopa vegyes feladattípusára. Végül megadtunk egy hatékony Monte Carlo integrálási eljárást a kétlépcsős modell várható pótlás függvénye értékeinek kiszámítására normális eloszlású jobboldali vektor esetére. Ez a módszer antitetikus változókat, iránymenti integrálást és rétegezett mintavételt alkalmaz, ezek együttes használata bizonyult hatékonynak.

Összegezve az eddigieket: az *SRA* algoritmus használatával lehetővé válik, hogy egyetlen numerikus optimalizálási eljárással kezeljük a sztochasztikus programozás két különböző feladattípusát, a valószínűségi feltételt és a várható pótlás függvényét együttesen is használhatjuk, továbbá kvadratikus célfüggvényt és/vagy kvadratikus feltételeket is beépíthetünk sztochasztikus programozási feladatokba – tehát az eddig ismert sztochasztikus lineáris programozási feladatok helyett áttérhetünk sztochasztikus kvadratikus programozási feladatok megoldására.

Megjegyezzük, hogy véleményünk szerint numerikus algoritmusokról és hatékonyságukról addig nem alakíthatunk ki megalapozott véleményt, amíg megfelelő számítógépes futásokkal nem mutattuk meg megvalósíthatóságukat és hatékonyságukat. A fentebb leírt, 3., 4., 5. és 6. fejezetekben megadott algoritmusokat nagyon sok numerikus példán lefuttattuk, és ezzel demonstráltuk a hatékonyságot. Az *SRA* algoritmus számítógépes futásaiból a disszertációban a következőket adtuk meg: a gyökkeresés zajos függvénye esetén 8 feladatot, a STABIL modell egy feladatának 8 változatát, a kétlépcsős feladat 5 kis és közepes méretű példájának összesen 18 változatát, a vegyes modell egy feladatának 8 változatát vizsgáltuk, valamint 124 kétlépcsős feladatot és ezek általunk meghatározott közelítő megoldásait a `www.math.bme.hu/~deak` címen közzétettük. A számítógépes megvalósítás folyamán mintegy 12000 sor Fortran nyelvű programot készítettünk, nem számítva a többször használt, vagy készen átvett részeket.

Köszönetnyilvánítás. Kedves kötelességemnek érzem, hogy köszönetet mondja Prékopa Andrásnak, aki már az egyetemi tanulmányaim alatt tanított, majd szerencsém volt az általa vezetett kutatócsoportokban dolgozni. Neki köszönhetem az operációkutatás érdekes témáiba való bevezetést, az ötleteket és a gyakorlati alkalmazásokat, továbbá egész kutatói habitusomat. Kutatásaim témáit is ő inspirálta. Szeretném megköszönni M.A.H. Dempster, P. Kall és S.M. Robinson tanár urak segítségét, akik külföldi munkavállalásaimat

segítették elő. □

1. fejezet

Véletlenszámgenerátorok

Minden szimulációs számítás (Monte Carlo módszer) valószínűségi változók független realizációit igényli. Számításainkban az általános gyakorlatnak megfelelően számítógépes programokkal (generátorokkal) állítunk elő (pszeudó)véletlen számokat. Ebben a fejezetben azt írjuk le, hogyan lehet nem egyenletes, diszkrét eloszlású véletlen számokat előállítani, kevés számú egyenletes szám segítségével [De 86a], illetőleg hogyan lehet párhuzamos számítógépeken nagymennyiségű egyenletes eloszlású számot generálni [De 90b].

1.1. Takarékos módszer diszkrét valószínűségi változók generálására

A nem egyenletes, diszkrét eloszlású véletlen számok generálásának egyik alapvető eszköze az elfogadás-elvetés módszere. Az egyik hátránya ennek az eljárásnak, hogy néhány (vagy sok) véletlen számot elvetünk, mielőtt egy elfogadott értéket elő tudunk állítani. A takarékos módszer [De 81] egy olyan generátort jelent, amelyben az elvetett véletlen számokat is felhasználjuk. Ebben a részben azt írjuk le, hogyan lehet a takarékos módszer alapötletét tetszőleges diszkrét eloszlásra alkalmazni.

1.1.1. Egy egyszerű példa

A takarékos módszer lényege az, hogy ha olyan intervallumban generálunk egy számot, amelyben már úgyis sok generált szám van (felesleg intervallum), akkor ezt a számot egy megfelelő transzformációval a hiány intervallumba visszük át.

Szemléltetésként bemutatjuk, hogyan működik ez az eljárás egy kétpontos tartójú η valószínűségi változó esetén. Legyen $P\{\eta = 1\} = p_1, P\{\eta = 2\} = p_2, p_1 > 0, p_2 = 1 - p_1 > 0, p_1 < p_2$. Ha egyenletesen generálnánk i-t az $\{1, 2\}$ halmazon, akkor az 1 index a felesleg, a 2 index pedig a hiány intervallumát jelenti (hiszen $p_1 < 1/2, p_2 > 1/2$), a felesleg indexből a hiány indexbe történő transzformáció $T(1) = 2$. Ennek megfelelően η valószínűségi változót állít elő a következő generátor.

Takarékos eljárás kétpontos diszkrétre.
1. Generáljuk az i indexet egyenletesen az $\{1,2\}$ halmazon.
2. Ha $i=2$ (hiány index) akkor adjuk át $i=2$-t.
3. Ha $i=1$, akkor véletlen teszt segítségével döntünk az index helybenhagyásáról vagy transzformálásáról: generáljuk v-t egyenletesen a $[0,1)$ intervallumban, ha $v \leq 2p_1$, akkor adjuk át i-t, egyébként transzformálunk: adjuk át $i=2$-t.

1.1.2. Az általános eset

Minden n pontos tartóval rendelkező diszkrét eloszlás felírható úgy, mint n darab két-pontos eloszlás egyenlő valószínűségekkel vett keveréke – ezt Walker [Wal 77] mutatta meg azzal kapcsolatban, hogy az általa „álneves" módszernek nevezett generátortípust bevezette (lásd még [KP 79]).

A módszer formális leírásához tekintsünk egy ξ valószínűségi változót, amely n pontra koncentrálódik, $P\{\xi = x_i\} = p_i, p_i > 0, i = 1, \ldots, n, p_1 + \cdots + p_n = 1$ és n darab η_i valószínűségi változót, amelyek két-pontos eloszlásúak: $P\{\eta_i = y_{ji}\} = q_{ji}, q_{ji} > 0, j = 1,2, i = 1, \ldots, n, q_{1i} + q_{2i} = 1$. Legyenek az η_i valószínűségi változók azok, amelyek eloszlásainak azonos súlyokkal vett összege éppen a ξ eloszlását adják. Az általánosság megsértése nélkül feltehetjük, hogy $q_{1i} \leq q_{2i}$. A $P\{\eta_i = y_{2i}\} = q_{2i} = 1$ alakú degenerált eloszlásokat transzformáljuk az egyenletes eloszlásba, vagyis legyen $y_{1i} = y_{2i}$ és az eloszlás új alakja $P\{\eta_i = y_{1i}\} = q_{1i} = 1/2$ és $P\{\eta_i = y_{2i}\} = q_{2i} = 1/2$.

Építsük fel az $\mathbf{r} = (r_1, \ldots, r_{2n})$ vektort a következőképpen. A $q_{1s} = q_{2s} = 1/2$ azonos értékű valószínűségeket helyezzük el a vektor közepén, az egy eloszláshoz tartozó, nem azonos értékű $q_{1s} < q_{2s}$ valószínűségeket pedig töltsük be az $\mathbf{r}$-be kívülről befelé haladva. Pontosabban definiáljuk az r_k, z_k mennyiségeket a következő algoritmussal.

Előkészítés
1. Legyenek $k=1, m=1, i=0$ a kezdeti értékek.
2. Növeljük meg a számlálót, legyen $i=i+1$. Ha $i>n$, akkor menjünk az 5. lépésre.
3. Ha $q_{1i}=q_{2i}$ akkor menjünk a 4. lépésre, egyébként legyen $r_k=q_{1i}$, $r_{2n-k+1}=q_{2i}, z_k=y_{1i}, z_{2n-k+1}=y_{2i}, k=k+1$ és menjünk vissza a 2. lépésre.
4. Legyen $z_{n-m+1}=y_{1i}, z_{n+m}=y_{2i}, m=m+1$ és menjünk vissza a 2. lépésre.
5. Tároljuk az $n^+=k, n^-=n+m$ értékeket.

Itt, és a további algoritmusok leírásában az $y = x$ egyenlőséget a FORTRAN nyelvben megszokott módon kell értelmezni, az x értékét betesszük az y változóba. Jegyezzük meg, hogy a generátor végső formájában nem lesz szükségünk az $r_{n^+}, \ldots, r_{2n}$ értékekre, elég azt tudnunk ezekről, hogy egyik sem kisebb 1/2-nél. Az $1, \ldots, n^+ - 1$ indexek a felesleg

indexek, a hiány indexek pedig az $n^-, \ldots, 2n$. Azok az értékek, amelyek valószínűségeinek indexei n^+ és $n^- - 1$ között vannak, a valószínűségek 1/2-el egyenlőek és hiány indexeknek tekinthetők. Egy i indexű felesleg érték transzformációja megfelelő hiány helyére egyszerűen a $2n - i + 1$ transzformáció.

A generálási eljárás ezek után a következő. Állítsuk elő a k indexet egyenletesen az $1, 2, \ldots, 2n$ közül. Ha $i \geq n^+$, akkor átadjuk z_k-t, egyébként egy véletlentől függő tesztet hajtunk végre: generáljuk u-t egyenletesen a [0,1) intervallumból, ha most $u < 2r_k$, akkor átadjuk a z_k értéket, mint a generált számot, egyébként pedig a transzformált z_{2n-k+1} értéket adjuk át. A generálási eljárásban egy minta előállításához szükséges egyenletes eloszlású minták számának várható értéke $\overline{N} = 1.5$, vagy ennél kisebb, ha $n^+ \leq n$. Hasonlítsuk ezt össze azzal a helyzettel, amikor az „álneves" módszerben egy egyenletes szám szükséges az eloszlás kiválasztására és egy másik kell az ebből az eloszlásból való mintavételhez, vagyis $\overline{N} = 2$.

A takarékos módszer pontos leírása felhasználja azt az általánosan használt eljárást is, amikor egy [0,1)-ben egyenletes eloszlású véletlen számot felosztunk egy egyenletes eloszlású indexre és egy [0,1)-ben egyenletes eloszlású u^* véletlen számra, ami által $\overline{N}$ értékét 1-re csökkentjük. Az algoritmus leírásában $[x]$ az x egész részét jelenti.

Takarékos algoritmus

1. Generáljuk u-t egyenletesen [0,1)-ben.
2. Számítsuk ki a $k = [2nu] + 1$ véletlen indexet.
3. Ha $k \geq n^+$, akkor adjuk át a z_k-t.
4. Egyébként számítsuk ki az $u^* = 2nu - k + 1$ számot, ha $u^* < 2r_k$, akkor adjuk át a z_k értéket, egyébként pedig adjuk át a transzformált z_{2n-k+1} értéket.

1.1.3. Aszimptotikus viselkedés

Elméleti szempontból érdekes megjegyezni, hogy egy tetszőleges eloszlású diszkrét valószínűségi változó előállításához szükséges egyenletes minták száma aszimptotikusan 1-re redukálható a fentebbi ötlet ismételt alkalmazásával. A kérdés csak elméleti érdekességű, hiszen a memóriaigény exponenciálisan nő és az egyenletes számnak egy indexre és egy maradék egyenletesre való felbontása már eleve csak egy egyenletes eloszlású számot igényel a gyakorlatban.

Ismételten alkalmazzuk az előző ötletet a fentebbi $2n$ komponensű vektorra a következő értelemben. Cseréljük le az $\mathbf{r}$ vektort a következő $4n$ elemű $\mathbf{s} = (s_1, \ldots, s_{4n})$ vektorral: legyen $s_i = 2r_i, s_{4n-i+1} = 2(r_{2n-i} - 1/2), i = 1, \ldots, n^+ - 1$, ahol feltesszük, hogy $s_i < s_{4n-i+1}$ (egyébként felcseréljük a két értéket). A megmaradó $s_i, i = n^+, n^+ + 1, \ldots, 4n - n^+$ valószínűségek pedig mind egyenlőek 1/2-del. Ekkor az $\mathbf{s}$ vektor és a megfelelő generátor hasonló az előző $\mathbf{r}$ vektorhoz és a T generátorhoz, azzal a különbséggel, hogy legalább $2n$ valószínűség egyenlő 1/2-del. Az eredeti n pontos eloszlást előállítottuk mint a $2n$ darab

(s_i, s_{4n-i+1}) kétpontos eloszlás azonos súlyokkal vett keverékét.

Ha ezt a felbontást k-szor megismételjük, akkor a szereplő valószínűségek legalább $1 - \frac{2}{2^k}$-ad része egyenlő lesz 1/2-del, tehát egy olyan automatikus eljárást állítottunk elő, amely (túlnyomó részben) egyenletes eloszlások keverékeként adja meg az eredeti n pontos eloszlást. Az ennek megfelelő generátorban pedig határértékben csak $\overline{N} = 1$ darab egyenletes eloszlású számra van szükség. Ez az eljárás egy példa arra, hogyan lehet „memórián időt venni".

1.2. Egyenletes eloszlású számok generálása párhuzamos számítógépeken

A szimulációs algoritmusok számítógépes megvalósítása során mindig felmerül az a fontos kérdés, hogy milyen generátort használjunk egyenletes eloszlású pszeudovéletlen számok előállítására. Az erre a kérdésre adott válasz meghatározza a felhasznált számok minőségét, és ezzel a végső eredmény minőségét is befolyásolja. Párhuzamos számítógépek használata esetén ez a kérdés még fontosabbá válik, mivel ilyenkor nagyon nagy mennyiségű (pszeudo)véletlen számot használunk fel. A jelen szakaszban a véletlenszám-generátor sok processzorral rendelkező számítógép esetén történő kiválasztásának és kezdeti beállításai meghatározásának szempontjait mérlegeljük. Röviden összefoglalva olyan eltolásos generátorokat ajánlunk párhuzamos számítógépeken való használatra, amelyek nagy Mersenne prímmel felírt primitív trinomiálokon alapszanak.

Az eltolásos generátorok (Feedback Shift Register sequences) használatát Tausworthe [Tau 62] javasolta véletlen számok generálására, ennek továbbfejlesztését, az általánosított eltolásos generátorokat lásd a [LP 73] cikkben. Marsaglia és Tsay [MT 85] megmutatták, hogy ennek a generátornak is vannak hibái, de ennek ellenére az alkalmazott, elég szigorú statisztikai teszteket kielégítik (Niederreiter [Nie 87] meghatározta a generált sorozatok diszkrepanciáját is).

A következő 1.2.1 pontban megtárgyaljuk az általánosan használt kongruenciális generátorok párhuzamos számítógépeken történő felhasználását, valamint az eltolásos generátorok előnyeit. A harmadik részben megadunk egy ekvivalenciát a generált számok és egy polinom-test között, majd a véletlenszám generátorok kezdeti értékeinek beállításához használt eljárásokat, az inicializálást írjuk le – itt felhasználjuk Collings és Hembree [CH 86] technikáit, de az általuk használttól eltérő módon. A negyedik részben a Thinking Machine Corporation Connection Machine számítógépének és a Floating Point Systems cég T-Series gépének jellemzőit ismertetjük vázlatosan [Fre 86]. Az utolsó két pontban azt tárgyaljuk meg, hogy ezen a két gépen (géptípuson) hogyan végezzük a véletlenszám-generátorok implementálását és inicializálását.

1.2.1. Egyenletes véletlenszám generátorok

A szimulációs számításokban szükséges egyenletes eloszlású valószínűségi változók független realizációinak előállítására a leggyakrabban a kongruenciális generátorokat használják. Ezek egy x_0 kezdeti értékből kiindulva az

$$x_{i+1} = bx_i + c \pmod{m}$$

rekurziót használva állítanak elő egy számsorozatot, ahol b, c, m adott értékek; az m modulus általában a gépben reprezentálható legnagyobb egész, vagy egy ennél némileg kisebb szám. Ebből az

$$u_i = x_i/m$$

transzformációval állítjuk elő a $[0, 1)$ intervallumban lévő számokat, melyeket a $[0, 1)$-ben egyenletes eloszlású valószínűségi változó független realizációiként használnak. A generátorok egyik legfontosabb tulajdonsága a periódus-hossz, amely mindig kisebb, mint az általában $m = 2^{32}$ értékű modulus. Ez a szám nagyon kicsinek tűnik, ha az ezer, vagy még több processzort használó párhuzamos számítógépekre gondolunk. Hasonlóképpen kicsinek tűnik ez a szám, ha többdimenziós integrálok kiszámításának, vagy optimalizálási feladatok megoldásának szükségleteivel vetjük össze, hiszen itt $10^{10} - 10^{15}$ számú véletlen pontra is szükség lehet. Többdimenziós integráloknál, illetőleg az n-dimenziós $(u_i, u_{i+1}, \ldots, u_{i+n-1})$ pontok esetében nem feledkezhetünk meg a rácsszerkezetükről sem, vagyis arról, hogy ezek a pontok elég kevés számú, egymással párhuzamos hipersíkon vannak.

Végül azt is figyelembe kell venni, hogy az összes lehetséges, rendelkezésünkre álló számnak csak kis részét használhatjuk fel, nehogy az óhatatlan mellékhatások számításaink megbízhatóságát lerontsák. Például említjük DeMatteis és Pagnutti [DP 88] cikkét, amelyben a kongruenciális generátor generálta számsorozat igen távoli tagjai közötti korrelációt írják le, és eredményeik alapján a felhasználható számoknak legfeljebb a 0.1%-át javasolják tényleges felhasználásra.

Véletlenszám generátoroknak a több (sok) processzorral dolgozó párhuzamos számítógépeken való alkalmazása esetén választanunk kell a következő két lehetséges implementálási mód között:

(i) ugyanazt a kongruenciális generátort használjuk minden processzoron és az általa előállított sorozat különböző szegmenseit használjuk a különböző processzorokon, vagy

(ii) minden egyes processzor esetén különböző kongruenciális generátort használunk.

Az első esetben előfordulhat, hogy még egész kis feladatok esetén is kimerítjük a felhasználható számokat munkánk befejezése előtt. A második esetben pedig arra lenne szükségünk, hogy a kongruenciális generátorok százait, vagy ezreit állítsuk elő (jó minőségű számsorozatok előállítását garantáló b, c, m állandókkal) – ami megint számos nehézséget jelentene.

Mindezeket figyelembe véve a szerző határozott véleménye, hogy a kongruenciális generátorok sem ma, sem a jövőben nem használhatók párhuzamos számítógépeken. Viszont

egy lehetséges kiutat találhatunk az eltolásos generátorok alkalmazásával. Ezeknek a használata röviden a következőképpen fogalmazható meg. Legyen adva az $x_0, x_1, \ldots, x_{p-1}$ kiindulási bitsorozat ($x_i = 0$ vagy 1), ekkor az $\{x_i\}$ bitsorozat további elemeit a

$$x_k = c_1 x_{k-1} + c_2 x_{k-2} + \cdots + c_p x_{k-p} \pmod 2$$

rekurzióval állítjuk elő, ahol $c_1, c_2, \ldots, c_p$ adott 0 vagy 1 értékű állandók. Az $\{x_i\}$ bitsorozat akkor éri el a maximálisan lehetséges 2^{p-1} periódushosszat, ha a generátornak megfelelő

$$f_0(x) = 1 + c_1 x + c_2 x^2 + \cdots + c_p x^p$$

polinom primitív a 0,1 együtthatós n-edfokú polinomok testjében, a GF(2)-ben. Az egyszerűség és a számítástechnikai hatékonyság kedvéért a továbbiakban csak primitív trinomiálokkal foglalkozunk, vagyis amikor a polinom alakja

$$f(x) = 1 + x^q + x^p, q \leq p/2.$$

Ennek a polinomnak megfelelő generátor a következő bit előállításához csak egy bitek közötti „kizáró vagy" művelet végrehajtását igényli, hiszen ezen polinom szerint a rekurzió formája

$$x_k = x_{k-p} + x_{k-p+q} \pmod 2,$$

pontosabban az eredmény a $x_k = x_{k-p} + x_{k-q} \pmod 2$ egyenlet lenne, de az általunk adott rekurzió ugyanazt a sorozatot generálja, fordított sorrendben. Egy, párhuzamos számítógépen használható, megfelelően nagy p kitevővel rendelkező primitív trinomiál nagyon nagy periódushosszal rendelkezik. Ilyen trinomiálok adatait az irodalomban meg lehet találni, például [LN 83], [De 90a]. Tehát az eltolásos generátorok használata legalábbis megengedett és a kis periódushossz miatti nehézségekre megoldást jelent.

További jó tulajdonsága az eltolásos generátoroknak, hogy ha a bitsorozatot felvágjuk például 32 bit hosszúságú darabokra (decimáljuk), akkor a kapott számsorozat a legfeljebb $(2^p-1)/32$ dimenziós terekben is teljes egyenletességet mutat [Fus 88]. Egy ilyen generátor párhuzamos gépeken való implementálása és inicializálása, valamint a véletlen számok generálása viszonylag könnyű, amint azt az alábbiakban megmutatjuk.

A következő eldöntendő kérdés, hogy milyen p és q értékeket válasszunk. Több okból is egy Mersenne prímet érdemes választani, vagyis egy olyan prímszámot, amely esetén még a $2^p - 1$ érték is prím. A Mersenne prím használata önmagában biztosítja, hogy bizonyos statisztikai teszteket a generátor által adott számsorozatok jó eredménnyel kielégítenek. Az ajánlott megvalósításban a fentebb felvetett (i) és (ii) alatti két lehetőség közül az elsőt fogjuk választani, vagyis ugyanazt a generátort használjuk minden processzoron, és az egyik processzoron használt számsorozat szegmenstől jóval távolabb választjuk a másik processzoron használt szegmens kezdőindexét – a két szegmensben használt bitek indexe közti eltérést eltolásnak, vagy ugrásnak nevezzük. Ilyenkor az a tény, hogy a $2^p - 1$ számnak nincs osztója, már önmagában azt jelenti, hogy az eltolt sorozat (vagy a későbbiekben

leírásra kerülő GFSR sorozatok) szintén megőrzi az eredeti, teljes periódushosszat. A p és q értékek megválasztásánál még azt is figyelembe vehetjük, hogy a q értéke ne legyen közel a $p/2$ értékhez és ne legyen nagyon kicsi se – ezen értékek kizárását bizonyos statisztikai tesztek eredményei alapján kívánjuk meg. Három lehetséges számpár a (p, q) értékek választására a következő: (521,48), (2281,715) és (9689,1863).

1.2.2. Polinomok és eltolásos sorozatok ekvivalenciája

Megadunk egy megfeleltetést a polinomok és az eltolásos sorozatok között, mindkettő véges testekben van. Itt az egy-egy megfeleltetés nyilvánvalóan igaz, de az alábbiakban megmutatjuk, hogy a megfeleltetés könnyen kiszámítható. Az állítások megfogalmazhatók általánosabban is, de nekünk az itt kimondott alakok elégségesek. Legyen

$$f(x) = 1 + x^q + x^p,\ q \leq p/2$$

egy primitív trinomiál. Egy $x_0, x_1, \ldots, x_{p-1}$ kezdeti szám p-esből a

$$x_i = x_{i-p} + x_{i-p+q}, \quad \mod 2 \tag{1.1}$$

rekurzióval kapott bitsorozatot jelölje $\{x_i\}$. A rekurzió k-szoros alkalmazásával kapjuk a

$$x_k, x_{k+1}, \ldots, x_{k+p-1} \tag{1.2}$$

szám p-est, amelyet a kezdeti vektor és a használt $f(x)$ trinomiál teljesen meghatároz. Tekintsük most az $a_i = 0$ vagy $a_i = 1$ együtthatókkal rendelkező

$$g(x) = a_0 + a_1 x + a_2 x^2 + \cdots + a_p x^{p-1}$$

polinomokat tartalmazó véges testet, ahol a szorzást és az összeadást az $f(x)$-re, mint modulusra végezzük el, illetőleg az együtthatókra nézve a mod 2 összeadást használjuk. Ezek a polinomok egy véges, 2^p elemet tartalmazó véges testet alkotnak, és a $2^p - 1$ nemzérus polinom egy ciklikus csoportot alkot, ahol x egy primitív elem, vagyis az $1, x, x^2, \ldots, x^{2^p-2}$ elemek mind különbözők. Így a

$$\begin{aligned} g_k(x) &= b_0 + b_1 x + b_2 x^2 + \cdots + b_p x^{p-1}, \\ g_{k+1}(x) &= x g_k(x), \\ g_{k+2}(x) &= x^2 g_k(x), \\ &\vdots \\ g_{k+p-1}(x) &= x^{p-1} g_k(x) \end{aligned} \tag{1.3}$$

polinomok is mind különbözőek. Vegyük észre, hogy ezek a polinomok ugyanazt a rekurziót elégítik ki, mint a véletlen bitek, hiszen

$$g_{k+p}(x) = x^p g_k(x) = (1+x^q)g_k(x) = g_k(x) + g_{k+q}(x)(\text{mod } f(x)). \tag{1.4}$$

Most megmutatjuk, hogy az (1.3) alakú polinomok és az (1.2) alakú bináris sorozat között a következő kongruenciák által lehet megfeleltetést megadni:

$$\begin{aligned} x_k &= g_k(x)|_{x=1}(\text{mod } f(x)), \\ x_{k+1} &= g_{k+1}(x)|_{x=1}(\text{mod } f(x)), \\ &\vdots \\ x_{k+p-1} &= g_{k+p-1}(x)|_{x=1}(\text{mod } f(x)), \end{aligned} \tag{1.5}$$

ahol az eredmények (mod $f(x)$) és (mod 2) számítandók. Mivel az $f(x)$ a $\{0,1\}$ kételemű test feletti polinomgyűrűben vannak, az $x = 1$ helyettesítés után szintén a (mod 2) szerint kell venni a végső eredményt.

Az (1.5) egyenletrendszer szerint a polinomsorozatból a bitsorozat könnyen meghatározható. A fordított irány esetére – bitsorozatból a polinomsorozat meghatározása – némi számításra van szükség. Az (1.3) egyenletek alapján a $j = 0, 1, 2, \ldots, p-q$ indexekre fennáll, hogy

$$\begin{aligned} x_{k+j} &\equiv g_{k+j}(x)|_{x=1} = x^j g_k(x)|_{x=1} \\ &= b_0 x^j + b_1 x^{j+1} + \cdots + b_{p-j-1}x^{p-1} + b_{p-j}x^p + b_{p-j+1}x^{p+1} + \\ &\quad \cdots + b_{p-1}x^{p+j-1}|_{x=1} \\ &\equiv b_0 x^j + b_1 x^{j+1} + \cdots + b_{p-j-1}x^{p-1} + b_{p-j}x^q + b_{p-j} + \\ &\quad b_{p-j+1}x^{q+1} + b_{p-j+1} + \cdots + b_{p-1}x^{q+j-1} + b_{p-1}x^{j-1}|_{x=1} \\ &= (b_0 + b_1 + \cdots + b_{p-1}) + (b_{p-j} + b_{p-j+1} + \cdots + b_{p-1}), \end{aligned}$$

amiből pedig

$$\begin{aligned} x_k &\equiv b_0 + b_1 + \cdots + b_{p-1}, \\ x_{k+j} &\equiv x_k + b_{p-j} + b_{p-j+1} + \cdots + b_{p-1}. \end{aligned}$$

Tehát $j = 0, 1, 2, \ldots, p-q$ esetén fennáll a

$$x_{k+j} \equiv x_{k+j-1} + b_{p-j}(\text{mod } 2) \tag{1.6}$$

rekurzió. Hasonlóképpen felírható a $j = p-q+1, p-q+2, \ldots, p-1$ indexekre, hogy

$$\begin{aligned} x_{k+j} &\equiv b_0 x^j + b_1 x^{j+1} + \cdots + b_{p-j-1}x^{p-1} + b_{p-j}x^q + b_{p-j} + b_{p-j+1}x^{q+1} \\ &\quad + b_{p-j+1}x + \cdots + b_{2p-j-q-1}x^{p-1} + b_{2p-j-q-1}x^{p-q-1} \\ &\quad + b_{2p-j-q}x^q + b_{2p-j-q} \\ &\quad + b_{2p-j-q}x^{p-q} + \cdots + b_{p-1}x^{j+2q-p-1} + b_{p-1}x^{j+q-p-1} + b_{p-1}x^{j-1}|_{x=1} \\ &= (b_0 + b_1 + \cdots + b_{p-1}) + (b_{2p-j-q} + b_{2p-j-q+1} + \cdots + b_{p-1}) \\ &\quad + (b_{p-j} + b_{p-j+1} + \cdots + b_{p-1}), \end{aligned}$$

amely röviden írva

$$x_{k+j} = x_{k+j-1} + b_{p-j} + b_{2p-j-q} (\text{mod } 2). \tag{1.7}$$

Tehát látható, hogy az $x_{k+1}, \ldots, x_{k+p-q}$ bitekre a növekedés mindig pontosan egy b_j együttható volt, mivel az x-el való szorzás eredményeként egy x^p tagot kaptunk, melyet az $1+x^q$ kifejezéssel helyettesítettünk. Az $x_{k+p-q+1}$ tagra (és az $\{x_i\}$ sorozat ezután következő tagjaira) a szorzás után két tagot kapunk, melyekben x^p szerepel. A (mod $f(x)$) felírt egyenletrendszer (x_i adott, a b_j együtthatók ismeretlenek) tehát a következő:

$$\begin{aligned}
x_{k+1} &= x_k + b_{p-1}, \\
x_{k+2} &= x_{k+1} + b_{p-2}, \\
&\vdots \\
x_{k+p-q} &= x_{k+p-q-1} + b_q, \\
x_{k+p-q+1} &= x_{k+p-q} + b_{q-1} + b_{p-1}, \\
x_{k+p-q+2} &= x_{k+p-q+1} + b_{q-2} + b_{p-2}, \\
&\vdots \\
x_{k+p-1} &= x_{k+p-2} + b_1 + b_{p-q+1}, \\
x_{k+p} &= x_{k+p-1} + b_0 + b_{p-q}.
\end{aligned}$$

Ez a rendszer pedig könnyen megoldható az ismeretlen b_j együtthatókra:

$$\begin{aligned}
b_{p-1} &= x_{k+1} + x_k, \\
&\vdots \\
b_q &= x_{k+p-q} + x_{k+p-q-1}, \\
b_{q-1} &= x_{k+p-q+1} + x_{k+p-q} + (x_{k+1} + x_k), \\
&\vdots \\
b_1 &= x_{k+p-1} + x_{k+p-2} + (x_{k+q-1} + x_{k+q-2}), \\
b_0 &= x_{k+p} + x_{k+p-1} + (x_{k+q} + x_{k+q-1}).
\end{aligned} \tag{1.8}$$

Ezen egy-egy megfeleltetés teszi lehetővé, hogy a Collings és Hembree által javasolt ugrást (a bitsorozat eltolását) viszonylag könnyen végrehajtsuk, akármilyen nagy is az ugrás.

Tegyük fel, hogy $N = 2^d p$ nagyságú ugrást akarunk végrehajtani, vagyis egy $S_k = (x_k, x_{k+1}, \ldots, x_{k+p-1})$ alakú szám p-esből el akarunk jutni a $S_{k+N} = (x_{k+N}, x_{k+1+N}, \ldots, x_{k+p-1+N})$ bitsorozathoz. Ekkor nem hajtjuk végre a bitekre vonatkozó rekurziót N-szer, hanem a

következőképpen járunk el. Meghatározzuk az S_k-beli x_k bithez tartozó $g_k(x)$ polinom $b_0, b_1, \ldots, b_{p-1}$ együtthatóit, az (1.8) egyenlet szerint. A $g_{k+2^d p}(x)$ polinom meghatározható a

$$g_{k+2^d p}(x) = x^{2^d p} g_k(x) = (x^p)^{2^d} g_k(x) = (1 + x^q)^{2^d} g_k(x) \tag{1.9}$$

egyenlőség alapján. Egy polinom négyzetreemelése (mod 2) és (mod $f(x)$) nagyon egyszerű, hiszen

$$g^2(x) = \left(a_0 + a_1 x + \cdots + a_{p-1} x^{p-1}\right)^2 \equiv a_0 + a_1 x^2 + \cdots + a_{p-1} x^{2p-2},$$

ezért az (1.9) jobboldalán lévő d számú négyzetreemelés könnyen elvégezhető. Természetesen minden alkalommal, amikor az x^p kifejezés megjelenik, helyettesítjük az $(1 + x^q)$ formával. Miután meghatároztuk az $g_{k+N}(x) = g_{k+2^d p}(x)$ polinomot, akkor az (1.5) alapján meghatározzuk a

$$x_{k+N+j} = x_{k+2^d p+j} = g_{k+2^d p+j}(x)|_{x=1} = x^j g_{k+2^d p}(x)|_{x=1} (\text{mod } f(x))$$

biteket, $j = 0, 1, 2, \ldots, p-1$ esetére. Egyetlen részlet van hátra; tegyük fel, hogy az ugrás nagysága nem $2^d p$ alakú, hanem tetszőleges r konstans. Ekkor megkeressük az

$$r = \left(2^{d_1} + 2^{d_2} + \cdots + 2^{d_s}\right) p + e$$

felbontást, ahol $e < p$ egész, és a $d_i, i = 1, \ldots, s$ számok mindegyikére végrehajtjuk a fentebbi eljárást. Természetesen igaz a következő összefüggés:

$$g_{k+r}(x) = x^e g_k(x) \prod_{i=1}^{s} (1 + x^q)^{2^{d_i}} .$$

1.2.3. A Connection Machine és a T-Series gépek

Ezt a két számítógépet írjuk le röviden az alábbiakban, mert ez a két gép a párhuzamos számítógépes fejlesztések között két, egymástól némileg különböző felépítést testesít meg. Mindkettő száznál több processzorral rendelkezik, így a vizsgálatuk más, a jövőben előállításra kerülő párhuzamos gépekkel kapcsolatosan is hasznos lehet.

A Connection Machine egy erősen osztott felépítésű, de ugyanakkor nagyon jó összeköttetésekkel is rendelkező architektúra: minden processzorának csak kis feladatokat kell végeznie, de minden processzor nagyon gyakran kommunikál a többi processzorral – akármelyik processzort vagy memóriaelemet elérheti. Másrészről a T-Series számítógépet úgy tervezték, hogy a szükséges numerikus számításokat lokálisan oldja meg egy processzoron, a saját lokális memóriájának a felhasználásával, és olyan ritkán kommunikáljon a többi processzorral, amennyire csak lehet. Némi egyszerűsítéssel azt mondhatjuk, hogy a Connection Machine erősen kapcsolt (tightly coupled), míg a T-Series gép lazán kapcsolt (loosely coupled).

A Thinking Machine Corporation cég által készített Connection Machine gépnek van valószínűleg a legtöbb processzora – a sebességét 6000 Mips-re teszik. A számítógépet különböző kiépítettségben gyártják, a 16K processzáló elemtől (PE) a 64K processzáló elemig terjed a skála. Egy processzáló elem csak nagyon kis munkát végez, mivel egy PE-ben csak egy egy-bites aritmetikai vagy logikai művelet végrehajtására alkalmas processzor és 4K bites lokális memória van. A processzorok 16-os csoportokba vannak osztva és ezek a csoportok egy 12 dimenziós hiperkocka csúcsain helyezkednek el a legnagyobb kiépítettségű konfigurációban. A lebegőpontos műveleteket speciális szoftverek segítségével tudja elvégezni: két 32 bites szám összeadása 750ns alatt végezhető el, a teljes számítási kapacitását 2 GFlops-ra becsülik.

A gépet elsősorban egész értékű aritmetikára tervezték (szövegkeresés archívumban) és nagyon jó a processzorok közötti kommunikáció – a teljes, 32Mbyte memóriát 30msec alatt képes rendezni. Megjegyezzük, hogy a gép utasításai között a processzorok közötti kommunikációra van egy SEND parancs, amelynek van egy olyan opciója is, hogy ha két üzenetet küldenek ugyanarra a processzorra, akkor az üzeneteken bitenkénti „kizáró vagy" műveletet hajt végre. A gépnek ez a tulajdonsága nagyon kényelmesnek tűnik abban az esetben, ha primitív trinomiálokon alapuló véletlenszám generálással foglalkozunk.

A Floating Point System cégtől származó T-Series gép szintén 1986-ban jelent meg a piacon. A csúcssebességét 2 GFlops – 4GFlops-ra becsülik. A processzáló elemek száma a különböző kiépítésekben 1K és 16K között van, egy PE itt egy 32 bites Transputert tartalmaz, valamint 1Mbyte lokális memóriát. A Transputer önmagában 16 MFlops műveleti sebességre képes. A Transputereket nyolcasával processzor modulokba szervezték, és a modulokat egy 7-11 dimenziós hiperkocka élei mentén kötötték össze. A teljesítményt növeli, hogy minden processzor modulhoz hozzárendeltek egy keménylemezes háttértárolót is, 256Mbyte kisegítő memóriával.

A két gép összehasonlításánál vegyük észre az eltérő vonásokat is. Míg a Connection Machine a számítási kapacitását és a munkát az összes processzorra szétosztja a jó összeköttetést biztosító kapcsoló hálózat segítségével, a T-Series gép a számítási erőt és a memóriát modulokban koncentrálja. Némi túlzással mondhatjuk, hogy a Connection Machine felfogható egy nagyon nagy, de feldarabolható és újra konfigurálható számítógépnek, a T-Series inkább egymással összeköttetésben álló kisebb számítógépek hálózatának tekinthető.

Ma egyre inkább terjed az a megoldás, hogy egy egyetemen, vagy környékén lévő kihasználatlan PC-k felesleges számítási kapacitását felhasználják egyfajta párhuzamos számítógépként – ilyen szempontból a T-Series ezeknek a lazán kapcsolt sokgépes konfigurációk (MIMD) előfutárának tekinthető.

1.2.4. Eltolásos sorozatok a Connection Machine számítógépen

Véletlenszám generátorok Connection Machine számítógépre való alkalmazásakor a nagy $p = 9689, q = 1836$ állandókkal rendelkező eltolásos sorozatok használatát ajánljuk – egy

generátor alkalmazását az összes processzorokra, úgy, hogy a különböző PE modulokra a generált sorozat különböző szegmenseit használjuk, ezeket a szegmenseket egymástól ugrások választják el.

Tegyük fel, hogy rendelkezésünkre áll egy kezdeti $S_r = (x_r, x_{r+1}, \ldots, x_{r+p-1})$ szám p-es, amelyet az 1.2.3 szakaszban leírtak szerint állíthatunk elő. A kezdeti r ugrás nagyságának meghatározására Lewis és Payne az $r = 5000p$ értéket javasolta, ha az $x_0 = x_1 = \cdots = x_{p-2} = 0, x_{p-1} = 1$ sorozatból indulunk ki, de a már leírt technikák miatt vehetjük akár a $10^{10} - 10^{20}$ nagyságúnak az r értékét. Rendeljük hozzá az S_r első p bitjét az első p processzorhoz, vagyis töltsük be az x_r bitet a 0. processzor memóriájába, az x_{r+1} bitet az 1. processzor memóriájába, stb. Hajtsunk végre most egy d nagyságú ugrást az $\{x_i\}$ sorozatban, ahol d értéke legalább $10^5 pN$ legyen – $N = 10^{30}$ esetén egy $d > 10^{40}$ ugrás elég biztonságosnak látszik. Itt N jelöli a számítások alatt az egy processzorban előreláthatólag felhasználandó egyenletes eloszlású számok mennyiségét, ami a ténylegesen használt véletlen számok száma szorozva egy 10^5 biztonsági tartalékkal, hogy az összes lehetséges számnak csak a 0.1%-át használjuk fel.

Generáljuk a leírt módon, a megfelelő polinomszorzásokkal az $S_{r+d} =$
$= (x_{r+d}, x_{r+d+1}, \ldots, x_{r+p+d-1})$ sorozatot és tároljuk ezeket a $p, p+1, \ldots, 2p-1$ sorszámú processzorok memóriájába, stb. Végül marad még $7p$-64K=2287 számú bit, amelyeket a generálás folyamán külön kell kezelni, ezeket az utolsó két vagy három processzorhoz tartozó memóriába helyezzük el.

Az $\{x_i\}$ sorozat generálási eljárása ugyanaz a p elemet tartalmazó processzor-csoportok esetén, kivéve a legutolsó csonka csoportot, ahol a maradék 2287 bit miatt további programozási munkára van szükség. Egyszerűség kedvéért az első p darab processzor által használandó algoritmust írjuk le, feltéve, hogy ezeknek a processzoroknak a memóriájába már be van töltve a kezdeti $S_0 = (x_0, x_1, \ldots, x_{p-1})$ sorozat.

Egyenletes generátor a Connection Machine-re - első p processzor esetén.

1. Hajtsuk végre a ,,kizáró vagy'' műveletet a $0, 1, \ldots, q-1$ és a $q, q+1, \ldots, 2q-1$ sorszámú processzorok memóriájában tárolt biteken - a művelet eredményét tegyük a $0, 1, \ldots, q-1$ processzorok megfelelő (a véletlen bitek tárolására szolgáló) memóriájába a SEND utasítás segítségével. Ez az eredmény a $(x_p, x_{p+1}, \ldots, x_{p+q-1})$ lecseréli az eddigi $(x_0, x_1, \ldots, x_{q-1})$ bitsorozatot.

2. Végezzük el a $q, q+1, \ldots, p-1$ és a $2q, 2q+1, \ldots, p-1, 0, 1, \ldots, q-1$ processzorok tartalmának ,,kizáró vagy'' műveletét, az eredményt tegyük a $q, q+1, \ldots, p-1$ processzorokba. Ez az eredmény $(x_{q+1}, \ldots, x_{2p-1})$, és ezzel lecseréltük az $(x_q, \ldots, x_{p-1})$ sorozatot. Vegyük észre, hogy az $(x_{2q}, \ldots, x_{p-1}, x_p, \ldots, x_{p+q-1})$ sorozatot két részből konstruáltuk, az első $(x_{2q}, \ldots, x_{p-1})$ rész a $2q, \ldots, p-1$ processzoroknál volt tárolva, és része

volt a régi $0, 1, \ldots, p-1$ sorozatnak, míg a második rész, a $(x_p, \ldots, x_{p+q-1})$ már egy újonnan (az 1. lépésben) generált részsorozat és a $0, 1, \ldots, q-1$ processzoroknál volt tárolva.

A két lépés végrehajtása után (ami lényegében csak két SEND művelet), az első p processzorban van tárolva a $x_{p+1}, \ldots, x_{2p-1}$ bitsorozat. Mind a 7 processzorcsoport esetére ugyanezt az eljárást használva – egyidejűleg, az összes processzornál új véletlen bitet generáltunk, mindössze két SEND utasítással és gyakorlatilag semmilyen memóriára nincs szükség.

A közvetlen megközelítés kiindulásként egy 64Kbit hosszú $\{x_i\}$ sorozatot generálna, minden processzorhoz egy bitet, a generálási eljárásban pedig (egymás után) hétszer kellene végrehajtani a sorozat megújítását, vagyis a generálás mintegy hétszer ennyi ideig tartana.

Vizsgáljuk meg most azt a problémakört, hogy az általunk adott eljárás a lehetséges számok hányad részét használná fel. Tegyük fel, hogy a kezdeti sorozathoz $r = 10^{20}$ eltolással jutunk, az egyes szegmensek közötti eltolás pedig legyen $d = 10^{40}$ (Lewis és Payne $5000p \sim 10^7$ és $100p \sim 10^6$ értékeket javasolt). A hét szegmens hossza $7p \sim 7 \cdot 10^4$, tehát összesen

$$10^{20} + 6 \cdot 10^{40} + 7 \cdot 10^4 < 10^{41}$$

számra lehet szükségünk, ami még mindig messze van a teljes $2^{9698} \sim 10^{3000}$ periódushossztól. Egy sokkal kisebb kitevőjű primitív trinomiál, például a $p = 521$ kitevős polinom esetén is a periódushossz $2^{521} \sim 10^{150}$ megfelelőnek látszik.

Egy másik szempontból is megvizsgáljuk az esetleg szükséges véletlen számok mennyiségét. Egy nap mintegy 10^{14} nanosecundumból áll, és feltéve azt, hogy minden processzornak minden nanosecundumban szüksége van egy új véletlenszámra a szükségletünket $10^{14}p \sim 10^{18}$-ra teszi. Hozzávéve ehhez a biztonsági korlátot, hogy a generálható számoknak mondjuk csak 10^5p-ed részét használjuk fel még mindig csak 10^{27} nagyságrendű, egymástól különböző számot jelent. Tehát még egy egész napon át folyó szimuláció sem meríti ki a lehetséges számok halmazát.

Még egyszer visszatérünk az inicializáció kérdéséhez. Egyes esetekben szükségünk lehet arra, hogy egy véletlenszerű eltolással indítsuk a generátort. Ilyen véletlen ugrás értéket könnyen előállíthatunk, például egy kisebb, $x^{32} + x^7 + 1$ generátor segítségével. Természetesen a szegmensek közötti, megadott d ugrás helyett a processzorok mind a hét csoportja számára is előállíthatunk véletlen ugrásokat, vagy pedig egy kezdeti sorozat bitjeinek Fushimi [Fus 88] által leírt megkeverése is lehetséges. Az ugrás nagyságát beállíthatjuk p-nek egy olyan többszörösére is, ami néhány 2 hatványból áll csak – ez az ugrás végrehajtását megkönnyíti.

1.2.5. Általánosított eltolásos sorozatok a T-Series gépen

A T-Series gép felépítése alapján az általánosított eltolásos generátorok ([LP 73], Generalized Feedback Shift Register sequences – GFSR) használatát javasoljuk, amelyet a $p = 2281, q = 715$ állandókkal meghatározott véletlenszám generátorból építünk fel. Itt is egyetlen véletlenszámgenerátor alkalmazását javasoljuk, és a generált számok sorozatának különböző szegmenseit rendeljük hozzá a különböző processzorokhoz.

Az előző szakaszokban leírt eltolásos generátor által adott $\{x_i\}$ bitsorozat elemeinek egy speciális elrendezése alkotja a GFSR sorozatokat, amelyek sokkal használhatóbbak lebegőpontos aritmetika esetén, mint az eredeti bitsorozat. Egy GFSR sorozat első p darab, L bit hosszúságú számát a következőképpen adjuk meg:

$$\begin{aligned} W_0 &= x_0 x_d x_{2d} \cdots x_{(L-1)d}, \\ W_1 &= x_1 x_{d+1} x_{2d+1} \cdots x_{(L-1)d+1}, \\ W_2 &= x_2 x_{d+2} x_{2d+2} \cdots x_{(L-1)d+2}, \\ &\vdots \\ W_{p-1} &= x_{p-1} x_{d+p-1} x_{2d+p-1} \cdots x_{(L-1)d+p-1}. \end{aligned} \tag{1.10}$$

Ezek az L hosszú bitsorozatok úgy értendők, mint egy [0,1)-ben lévő lebegőpontos szám bitjei, vagyis a véletlen szám $y_j = W_j \cdot 2^{-L}$. Egyszerűség kedvéért a részletes leírásban az $x_0, x_1, \ldots, x_{p-1}$ kiindulási sorozatot használtuk, a gyakorlatban természetesen egy r kezdeti ugrást kell használni, mint az előző pontban. A használt d ugrás nagyságát $100p$-nek, vagy nagyobbnak kell vennünk. A következő véletlen számot a $W_p = W_0 \oplus W_q$ adja meg, ahol $\oplus$ műveletet bitenkénti „kizáró vagy”-ként hajtjuk végre. Általában a rekurziót

$$W_{i+p} = W_i \oplus W_{i+q}, i = 0, 1, 2, \ldots \tag{1.11}$$

adja meg. Tároljuk a $W_0, W_1, \ldots, W_{p-1}$ szavakat az első Transputerhez tartozó memóriába és vezessünk be egy v értékű ugrást az egyes processzorok közötti eltolás értékeként – ennek néhány nagyságrenddel (például $10^5 p$-vel) nagyobbnak kell lennie az egy processzor által felhasználásra kerülő véletlen számok N számánál; legyen például $v = 10^{40}$. Tároljuk a $W_v, W_{v+1}, \ldots, W_{v+p-1}$ szavakat a második processzor memóriájában, a $W_{2v}, W_{2v+1}, \ldots, W_{2v+p-1}$ szavakat a harmadik processzor memóriájában, stb.

A generálási eljárást minden processzorban szimultán végrehajthatjuk az (1.11) rekurzió szerint, így csak egyetlen lépést igényel új véletlen számok előállítása (esetleg szükség szerint a decimális alakra történő konvertálást is elvégezzük). A $W_p, W_{p+1}, \ldots$ szavak azoknál a processzoroknál kerül tárolásra, amelyeknél előállítottuk, tehát nincs szükség processzorok közötti kommunikációra. Természetesen az új szavakkal lecseréljük a régi, már nem szükséges szavakat, vagyis W_p lecseréli a W_0 tartalmát, a W_{p+1}-et a W_1 helyére tesszük, stb. így a memóriaigény nem nő.

A memóriaszükségletet a következőképpen lehet meghatározni. Minden processzornál p szót tárolunk, tehát esetünkben ez 8Kbyte memóriaigényt jelent (az elérhető 1Mbyte memóriából). Ha a program többi része nagyon memóriaigényes, akkor a $p = 521, q = 158$ választással ez lecsökkenthető 2Kbyte memóriaigényre. Egy további lehetőség a memóriaigény csökkentésére az, ha kihasználjuk, hogy a processzorok nyolcas modulokba vannak szervezve – egy modulból csak egy processzornál tároljuk a szükséges adatokat, a generálási lépésben ebben az egy processzorban nyolc új számot állítunk elő, és ezeket küldjük szét az azonos modulban lévő processzorokhoz. Ez a megoldás azért használható, mert a Transputer processzornak négy darab, kétirányú aszinkron csatornája van a kommunikációhoz, tehát ez a szétküldés nem terheli túlságosan a működést.

Az első processzornál felhasználandó bitek száma $n_1 = r + 32p + 31d \sim 5000p + 32p + 31 \cdot 100p < 10^4 p < 10^8$. A többi processzoroknál ez a szám kisebb, de tegyük fel, hogy ugyanennyi bitet használnak: $n_i = n_1 \sim 10^8 = n$. Az inicializáció során szükséges szavak száma $M = 16K$ processzor esetén $M(n + v) = 16K(10^8 + 10^{40}) < 10^{45}$. Hasonlítsuk össze ezt a lehetséges felhasználási számot a $2^{2281} \sim 10^{600}$ periódushosszal (vagy a kisebb kitevőjű primitív trinomiál $2^{521} \sim 10^{150}$ periódushosszával), és látható az ajánlott módszer megfelelősége. □

2. fejezet

Iteráció szemcsézettségű optimalizálás párhuzamos számítógépeken

Párhuzamosításnak nevezzük azt az eljárást, amelyben egy szekvenciálisan végrehajtandó lépéseket tartalmazó algoritmusból egy párhuzamos számítógépen használható változatot állítunk elő. Az ezzel kapcsolatos kutatások egyik részterülete az optimális szemcsézettség meghatározása, vagyis azon aritmetikai vagy logikai műveletek mennyisége, amelyet ugyanaz a processzor hajt végre – más processzorokkal való kommunikálás nélkül. A kis szemcsézettségű algoritmusok sok kommunikációt igényelnek, és általában valamilyen, processzorok közötti szikronizálásra is szükség van, míg a nagy szemcsézettségű eljárások használata folyamán csak ritkán van szükség kommunikációra és csak nagyon kismértékű szinkronizálás szükséges.

A szekvenciális optimalizálási módszerek párhuzamosítása esetén két, egymásnak ellentmondó követelményt szeretnénk egyszerre kielégíteni. Egyrészt a szinkronizációs költségek (szűk átviteli kapacitás, kommunikációs hálózat leterhelése, idő, várakozás más processzorok eredményeire) csökkentése és a kommunikációs szűk keresztmetszetektől való függetlenítés alapján minél kisebb kommunikációs költséget (komplexitás) szeretnénk elérni. Másrészt a jelentős gyorsulás elérése és a processzorok minél teljesebb kihasználása alapján a gyümölcsöző együttműködés a processzorok közti nagymennyiségű információcserét kívánja meg. Röviden összefoglalva az együttműködés lelassítja a végrehajtást, de együttműködés nélkül nem tudjuk kihasználni a sok processzorból adódó párhuzamosságot. Ha nincsen egy, a feladat természete által sugallt dekompozíció, akkor általában nagyon nehéz megmondani, hogy melyik az a nyeregpont, amelyben egy mindkét követelmény számára elfogadható egyensúly előállítható.

Ebben a szakaszban optimalizálási módszerek párhuzamosításának általános technikáját vizsgáljuk meg, amely elsősorban MIMD (Multiple Instructions Multiple Data Stream) típusú számítógépeken használható. Ezek közül is főként a lazán kapcsolt felépítésű, sok processzort használó párhuzamos számítógépekre alkalmazható a javasolt párhuzamosítás, mint például az Intel Hypercube architektúrájához hasonló felépítésű gépekre, vagy munkaállomások egy csoportjára (a szorosan kapcsolt számítógép struktúrák esetére kevésbé használ-

ható elgondolásunk, a Connection Machine típusú gépekre egyáltalán nem ajánljuk az eljárást.) Ezeknek a géptípusoknak a kommunikációs komplexitása igen kicsi és az átviteli hálózat is viszonylag lassú lehet. Ezek a tulajdonságok teljesen megfelelőek a javasolt globális dekompozíciós eljárásunk szempontjából, míg ugyanerre a géptípusra az iterációnál kisebb szemcsézettségű párhuzamosítás – a kommunikációs nehézségek miatt – gyakorlatilag használhatatlan.

Az általános számítási eljárásunkban a legjobb algoritmus automatikusan kiválasztódik, míg a többi algoritmus csak kiegészítő szerepet játszik. Legfőbb jó tulajdonsága a javaslatunknak a rugalmassága – a processzorok számától és az összekötő hálózattól szinte függetlenül (teljesen aszinkron módban) működik az így párhuzamosított eljárás.

2.1. A párhuzamosítás lehetőségei

A párhuzamosítás gyakorlatilag a megoldandó feladat dekompozícióját jelenti és némileg önkényesen az alkalmazásokat két különböző csoportba lehet osztani.

A feladatok nagy részét geometriailag lehet dekomponálni, vagyis azt a tartományt, amelyben a feladatot meghatározzuk, résztartományokra lehet bontani, és minden egyes processzor csak az egy résztartományra vonatkozó megoldás keresésével foglalkozik. Ezt az eljárást követik meteorológiai feladatok megoldásánál, vagy szélcsatorna kisérletek számítógépes szimulálásánál, ahol a résztartományok szerkezete a fizikai környezet által adott módon állítható elő. Hálózati folyamok és gráfelméleti feladatok megoldásánál is általában ilyen dekompozíciót használunk. A véletlen keresés algoritmusai (különösen a sok lokális minimummal rendelkező függvények vizsgálata) szintén ide sorolható – a változó értelmezési tartományát bontjuk fel sok kis részre. Ezt a megközelítést tartomány dekompozíciónak nevezzük.

A másik párhuzamosítási lehetőséget feladat dekompozíciónak nevezzük, ezek esetében a feladat matematikai modelljének a szerkezete sugallja a dekompozíciót. Ezt az utat követjük, ha a modell dekomponálható, mint például a legtöbb olyan lineáris algebrai feladat esetén, ahol a mátrixok blokkos szerkezetűek, vagy a korlátozás és szétválasztás módszerének alkalmazásában, amikor különböző ágakat különböző processzorokhoz rendelünk hozzá. A feladat természetéből fakadóan feladat dekompozíciót használunk, amikor olyan számítás-intenzív függvény is szerepel az optimalizálási feladatban, amely esetén a számítások elvégzése egymástól független részekre bontható, vagy egy függvény gradiensének kiszámítása során, amikor a gradiens egyes komponensei egymástól függetlenül meghatározhatók. Jegyezzük meg, hogy a feladat dekompozícióhoz a feladat alapos és mély megértésére van szükség, és mind a feladat, mind a módszer speciális tulajdonságait ki kell használnunk egy megfelelő párhuzamosítás előállításához. A megfelelő hatékonyság eléréséhez ezeken kívül még a teljes algoritmust hozzá kell igazítani a felhasználni kívánt párhuzamos számítógép szerkezetéhez és a kommunikációs hálózatához. Ez a dekompozíció sokszor a feladatban szereplő változó n dimenziójától függ és ezt sokszor nehéz összehangolni a felhasználható

processzorok N számával. Ez a kérdéskör akkor válik igazán fontossá, ha változó számú processzor áll a rendelkezésünkre.

Optimalizálási algoritmusok párhuzamosítása területén Bertsekas és Tsitsiklis [BT 89] könyvét, valamint Zenios [Zen 89] bibliográfiáját említjük csak meg (további hivatkozások a [LR 88], [BD 93], [De 96], [De 97b], [Pra 91] cikkekben találhatók). A párhuzamos optimalizálás területén eddig végzett kutatások és feladatmegoldások vagy tartomány dekompozíciós eljárások, vagy – kisebb mértékben – probléma-dekompozíciók kis szemcsézettséggel: az egy iteráción belüli műveleteket részekre osztják és egy ilyen részt egy processzorhoz rendelnek hozzá. A szerző azon a véleményen van, hogy mivel az optimalizálási algoritmusok túlnyomórészt iterációkon keresztül haladva érik el az optimumot, ezért a párhuzamos optimalizálás természetes szemcsenagyságának az iterációt kell tekintenünk. Az alábbiakban egy olyan módszert írunk le, amelyekben iterációkat rendelünk hozzá a processzorokhoz, és ezeket iteráció szemcsézettségű algoritmusoknak nevezzük.

A következő szakaszban ismertetünk egy eljárást, amelyet Bertsekas és Tsitsiklis [BT 89] dolgoztak ki, ennek során vezették be a teljes és részleges aszinkronitás fogalmát. Az általuk használt párhuzamosítás lényegében feladat dekompozíciós eljárás és komponensenkénti dekompozíciónak nevezzük, mert azon az erős feltevésen alapszik, hogy a megoldás komponenseit egymástól függetlenül lehet kiszámítani.

A 3. szakaszban egy másmilyen megközelítést alkalmazunk optimalizálási algoritmusok párhuzamosítására, amelyet globális dekompozíciónak nevezünk – a koordinátánkénti kiszámíthatóság elég ritkán teljesülő feltevése elhagyhatóvá válik, de az aszinkronitás megőrizhető, továbbá a megoldásvektorok n dimenziója és az N processzorszám egymástól teljesen független is lehet. Ennek az eljárásnak egy igen speciális esetét tárgyaltuk a [De 89] cikkben, egy másik egyszerű változatát pedig a [De 96] cikkben közöltük, az itt közölt általános leírás ezeket az egyszerűbb eseteket magában foglalja, további lehetőségeket a [De 97b] cikkben írunk le. A javasolt általános eljárás (igen kevés feltétel melletti) konvergenciáját is megmutatjuk. A következő szakaszban az általános modell előnyeit és hátrányait tárgyaljuk meg, míg az utolsó szakaszban a globális dekompozíciós eljárás lehetséges változatait adjuk meg.

2.2. Optimalizálás komponensenkénti dekompozícióval

A párhuzamosítás akkor végezhető el ennek a dekompozíciónak a segítségével, ha lényegében az egész optimalizálási feladat és a megoldási módszer mint a komponensek Descartes szorzata állítható elő – az egy komponensre vonatkozó számítási munkát egy processzorhoz (vagy egy processzorcsoporthoz) rendeljük hozzá.

Legyen a megoldandó optimalizálási feladat az F leképezés egy fixpontjának meghatározása, ahol F az optimalizálási eljárás egy iterációs lépésének tekinthető. Másszóval keressük azt az $\mathbf{x}^* \in X$ elemet, amelyre

$$\mathbf{x}^* = F(\mathbf{x}^*), \tag{2.1}$$

ahol $X = X_1 \times \cdots \times X_n$ valamilyen $X_1, \ldots, X_n$ halmazokkal, ahol $\mathbf{x} \in X$, $\mathbf{x} = (x_1, \ldots, x_n), x_i \in X_i, i = 1, \ldots, n$, továbbá $F : X \to X, F(\mathbf{x}) = (f_1(\mathbf{x}), \ldots, f_n(\mathbf{x}))$, $\forall \mathbf{x} \in X$. Az x_i értékek optimalizálási lépései koordinátánként hajthatók végre az $x_i := f_i(\mathbf{x})$ egyenletek alapján. Ha $x_i(t)$ a megoldásvektor i-edik komponensének értékét jelöli a t időpillanatban, akkor az iterációs lépés végrehajtását P_i processzornál az $x_i(t+1) := f_i(\mathbf{x}(t))$ egyenlet szolgáltatja és ezt a megoldást a többi processzor is eléri (ezt általában az $x_i(t+1)$ érték leadásával, minden processzorhoz való elküldésével érik el).

Az aszinkronitást a következő jelölések segítségével fogalmazzuk meg. Legyen T a $\{0, 1, 2, \ldots\}$ időpontok egy olyan részhalmaza, amikor az $\mathbf{x}$ megoldásvektor egy vagy több komponensét felfrissítjük bizonyos processzorok segítségével. Jelölje T^i azon időpontok halmazát, amikor az x_i értékét újraszámítjuk a P_i processzornál, vagyis $T = \cup_i T^i$. Nem biztos, hogy a t időpillanatban a P_i processzor hozzáfér a megoldás komponenseinek legfrissebb értékeihez (még nem érkezett meg a kiszámolt új érték), ezért a felfrissítést az

$$\begin{aligned} x_i(t+1) &= f_i\left(x_1(\tau_1^i(t)), \ldots, x_n(\tau_n^i(t))\right), \text{ ha } \forall t \in T^i, \\ x_i(t+1) &= x_i(t), \text{ ha } t \notin T^i, \end{aligned} \tag{2.2}$$

egyenletek írják le, ahol a $\tau_j^i(t)$ függvény egy időpillanatot határoz meg, amelyre fennáll a $0 \le \tau_j^i(t) \le t, \forall t$ egyenlőtlenség. A $t - \tau_j^i(t)$ különbség úgy tekinthető, mint P_j processzorból a P_i processzorba elküldött üzenet kommunikációs késése. Ily módon ha a P_i processzornál a t időpillanatban a P_j processzornál kiszámított eredményeket akarjuk használni, akkor az éppen aktuális $x_j(t)$ érték helyett csak a $x_j(\tau_j^i(t))$ érték áll a rendelkezésünkre a P_i processzornál.

Egy MIMD számítógépre azt mondjuk, hogy teljesen aszinkron módban működik, ha fennáll a következő feltevés.

1. Feltevés. *A teljes aszinkronitás TA feltétele*
i.) a T^i halmazok végtelenek és
ii) ha $t^i_{k_1}, t^i_{k_2}, t^i_{k_3}, \ldots$ egy olyan részsorozata a T^i elemeinek, melyre $\lim_{l\to\infty} t^i_{k_l} = \infty$ is fennáll, akkor $\lim_{l\to\infty} \tau_j^i(t^i_{k_l}) = \infty$ fennáll, minden $j = 1, 2, \ldots, N$ index esetén .

A feltevés első része azt biztosítja, hogy az x_i komponens értékét végtelen gyakran újraszámoljuk (a felfrissítések időpontjai nem korlátosak), a második rész pedig azt írja elő, hogy akármikor küldünk el egy üzenetet, az előbb, vagy később megérkezik a rendeltetési helyére. Másszóval mondhatjuk, hogy tetszőleges t_1 időponthoz létezik egy olyan későbbi időpillanat $t_2 > t_1$, amelyre $\tau_j^i(t) > t_1$ ha $t > t_2, \forall i, j \in \{1, 2, \ldots, N\}$ fennáll, vagyis a t_2 időpontig biztosan megérkezik a t_1-ig kiszámított eredmény.

A továbbiakban szükségünk van arra is, hogy az F optimalizálási eljárás (közönséges értelemben vett, szekvenciális használat esetén meglévő) konvergenciáját is biztosítsuk. Ezért használjuk a

2. Feltevés. *A szinkronizált konvergencia SC feltétele*
Létezik a nemüres $\{X(k)\}, k = 0, 1, \ldots,$ *halmazoknak egy olyan sorozata (amelyeket nivóhalmazoknak tekinthetünk) a következő tulajdonságokkal:*
(i) $\cdots \subset X(k+1) \subset X(k) \subset \cdots \subset X,$
(ii) $F(\mathbf{x}) \in X(k+1), \forall k, \mathbf{x} \in X(k),$
(iii) ha $\{\mathbf{y}_k\}$ *egy pontsorozat, melynek tagjai a nívóhalmazokból származnak,* $\mathbf{y}_k \in X(k), \forall k,$ *akkor az* $\{\mathbf{y}_k\}$ *sorozat minden torlódási pontja az* F *leképezés fixpontja.*

A Descartes-szorzat jellegű tulajdonságot nem csak az egyes komponensek újraszámításához kívánjuk meg, hanem az $X(k)$ halmazoknak is eleget kell tenniük egy hasonló kikötésnek:

3. Feltevés. *A doboz előírás BC feltétele*
Minden k *esetén léteznek olyan* $X_i(k) \subset X_i$ *halmazok, amelyekre* $X(k) = X_1(k) \times X_2(k) \times \cdots \times X_n(k).$

A feltételek leírása után meg tudjuk adni a következő, komponensenkénti dekompozícióra vonatkozó konvergencia-eredményt [BT 89]:

4. Tétel. *(komponensenkénti dekompozíció – aszinkron konvergencia). Ha fennállnak a feladat Descartes szorzat jellegére vonatkozó feltételek, vagyis a teljes aszinkronitás TA feltevése, a szinkronizált konvergencia SC feltevése és a doboz előírás BC feltevése, továbbá a kezdeti* $\mathbf{x}(0) = (x_1(0), \ldots, x_n(0))$ *megoldásra* $\mathbf{x}(0) \in X(0)$ *teljesül, akkor a* $\{\mathbf{x}(t)\}$ *pontsorozat minden torlódási pontja az* F *leképezés fixpontja.*

2.3. Az iteráció szemcsézettségű optimalizálási algoritmusok globális dekompozíciója

A szekvenciális optimalizálási algoritmusok konvergenciájának bizonyítását a Zangwill [Zan 69] által kifejlesztett elméleten belül lehet elvégezni. Ehhez az optimalizálási algoritmus egy iterációs lépését, mint egy $A : X \to Y$ pont–halmaz leképezést tekintünk. Az idevonatkozó eredmények ismertetését főleg Luenberger [Lue 84] jelöléseivel és kisebb mértékben Bazaraa [BSS 94] leírását követve adjuk meg.

Egy algoritmikus lépés (iteráció) úgy fogható fel, mint egy A leképezés, amely az előző iteráció végén kapott $\mathbf{x} \in X$ pontból előállít egy $A(\mathbf{x}) \in Y$ halmazt. Ha az A pont-halmaz leképezés zárt, akkor az A algoritmus ismételt alkalmazásával egy $\{\mathbf{x}_k\}, \mathbf{x}_{k+1} \in A(\mathbf{x}_k)$ pontsorozatot tudunk előállítani, amelynek a konvergens részsorozatai egy optimális megoldáshoz konvergálnak. A továbbiakban az ilyen algoritmusokat egyszerűen konvergensnek nevezzük. Egy további, enyhe feltétel esetén két konvergens $B : X \to Y, C : Y \to Z$ algoritmus egymásutáni alkalmazásával kapott $A = CB, A : X \to Z$ összetett algoritmus is konvergens lesz.

Néhány jelölést vezetünk be. Tegyük fel, hogy a

$$\min\ g(\mathbf{x}),\quad \mathbf{x} \in S \tag{2.3}$$

optimalizálási feladat megoldását keressük, ahol S egy konvex halmaz. Jelölje az optimális megoldások halmazát $\Gamma \subset S$, és tegyük fel, hogy létezik az S halmazon egy, az A, S, Γ függvényeként megadható Z leszálló függvény, amely rendelkezik a következő tulajdonsággal: ha az $A : S \to S$ leképezés egy konvergens algoritmust valósít meg, akkor $Z(\mathbf{y}) < Z(\mathbf{x}), \forall \mathbf{y} \in A(\mathbf{x})$, $\mathbf{x} \notin \Gamma$ esetén és $Z(\mathbf{y}) \leq Z(\mathbf{x}),\ \forall \mathbf{y} \in A(\mathbf{x})$, ha $\mathbf{x} \in \Gamma$ fennáll. Általában a leszálló függvény szerepére a (minimalizálandó) célfüggvényt használják, de néha ettől eltérő leszálló függvény konstruálására is szükség lehet. Egy B algoritmust az $\mathbf{y}_k$, k-adik iterációban előállított megengedett megoldás és a Z leszálló függvény tekintetében leszálló algoritmusnak nevezünk, ha az nem növeli meg a leszálló függvény értékét, vagyis ha $\mathbf{y}_{k+1} \in B(\mathbf{y}_k)$, akkor $Z(\mathbf{y}_{k+1}) \leq Z(\mathbf{y}_k)$ is fennáll, minden $\mathbf{y}_k \in S$ esetén.

A távolságtartó lépések tétele (spacer step theorem) fontos szerepet játszik a továbbiakban (a tételeket nem a legáltalánosabb formában mondjuk ki, csak a nekünk elégséges alakokat adjuk meg). Ez a tétel lényegében azt mondja ki, hogy ha egy H leszálló algoritmus egy (vagy több) lépését beillesztjük egy konvergens C algoritmus lépései közé, akkor az $A = CH$ (vagy $A = CH \cdots H$) összetett leképezés még mindig konvergens marad. Jegyezzük meg, hogy itt a H algoritmus lehet heurisztikus algoritmus is, nem okvetlenül csak konvergens algoritmus használható ebben a szerepkörben. Más megfogalmazásban ez azt jelenti, hogy ha egy konvergens algoritmus (távolságtartó) lépéseit időnként (de végtelen sokszor) beillesztjük egy heurisztikus leszálló algoritmus lépései közé, akkor az összetett leképezés konvergens lesz.

5. Tétel. *(Távolságtartó lépések tétele). Tegyük fel, hogy C egy konvergens algoritmus az S halmazon és hogy létezik egy C, S, Γ esetén használható Z leszálló függvény. Tegyük fel továbbá, hogy*
(i) rendelkezésünkre áll egy olyan $\{\mathbf{x}_k\}_{k=1}^{\infty}$ pontsorozat, amelyre $Z(\mathbf{x}_{k+1}) \leq Z(\mathbf{x}_k)$ fennáll, minden $k = 1, 2, \ldots$ esetén,
(ii) az $\{\mathbf{x} | Z(\mathbf{x}) \leq Z(\mathbf{x}_0)\}$ nívóhalmaz kompakt,
(iii) ha $k \in \mathcal{K}$, akkor $\mathbf{x}_{k+1} \in C(\mathbf{x}_k)$, ahol $\mathcal{K}$ egy végtelen indexhalmaz.
Ekkor az $\{\mathbf{x}_k\}, k \in \mathcal{K}$ tetszőleges konvergens részsorozatának az $\overline{\mathbf{x}}$ határértéke egy optimális megoldás, vagyis $\overline{\mathbf{x}} \in \Gamma$.

Ezen előkészületek után az optimalizálási algoritmusok iterációs szemcsézettségű párhuzamosítására ajánlott globális dekompozíciót, valamint a megfelelő számítási eljárást adjuk meg. A $P_i, i = 1, \ldots, N$ processzorokhoz hozzárendeljük a C_i és a H_i optimalizálási algoritmusokat, ahol a C_i algoritmusokról feltesszük, hogy konvergens eljárások, míg a H_i algoritmusról csak azt tesszük fel, hogy leszálló algoritmus minden olyan megengedett megoldás esetén, amelyet akármelyik processzornál a t időpillanatig kiszámítottunk (ezek a H_i algoritmusok lehetnek akár konvergens, akár heurisztikus algoritmusok is, csak a leszálló

természetüket kötjük ki). Az algoritmusoknak a processzorokhoz való hozzárendelése tetszőleges, de a kiválasztások elvégzése után nem változtatható. A C_i algoritmusok által adott megoldásokat $\mathbf{x}$, a H_i által kiszámított megoldásokat $\mathbf{y}$ jelöli. A H_i algoritmus k-adik lépéseként használhatjuk a következő, egyszerű kiválasztási szabályt:

$$\mathbf{y}_k^i = \mathbf{y}^i(t) = \text{argmin}_{m=1,2,\ldots,N}\ g(\mathbf{x}^m(\tau_j^i(t))), \tag{2.4}$$

amely nyilván egy leszálló algoritmust ad. A $\mathbf{x}^i(t)$ jelölést azért használjuk, hogy az időtől való függést is figyelembe tudjuk venni; ez az $\mathbf{x}^i(t)$ a P_i processzornál a t időpontban rendelkezésünkre álló megoldás, a $\tau_j^i(t)$ függvényt azt előző szakaszban definiáltuk, de most T^i csak azokat az időpillanatokat tartalmazza, amikor az $\mathbf{x}_k = \mathbf{x}^i(t)$ értékének kiszámítását befejeztük, vagyis a C_i konvergens algoritmikus lépés befejeződött.

A processzorok végezte feladatok és a kommunikációs hálózat működése egyszerűen megadható: a P_i processzor végrehajt egy C_i algoritmikus lépést, aztán a kapott $\mathbf{x}_k^i = \mathbf{x}^i(t)$ megoldást és az ehhez tartozó $g(\mathbf{x}^i(t)), k$ skalárokat (a célfüggvény értékét és az iterációs lépésszámot) szétküldi a többi processzornak (vagy feltesszük, hogy valamilyen módon az összes processzor hozzáfér a többi processzoroknál kiszámított $\mathbf{x}^i$,$g(\mathbf{x}^i)$,k értékekhez). A megértés elősegítése céljából tegyük fel, hogy a P_i processzornál van egy kijelölt munkaterület, amelyet $\mathbf{X}^i = \{X_m^i\}_{m=1}^N$ tömbként ($N \times (n+2)$ mátrixként) jelölünk, amelynek m-edik sora tartalmazza a P_m processzortól kapott eredményeket, az $X_m^i = (\mathbf{x}^m(\tau(t)), g(\mathbf{x}^m(\tau(t))), k)$ értékeket.

Miután a P_i processzor szétküldte az $\{\mathbf{x}^i(t), g(\mathbf{x}^i(t)), k\}$ értékeket, végrehajt egy H_i iterációs lépést az eddig kapott $\mathbf{X}^i$ megoldásokon, kiszámítva egy $\mathbf{y}_k \in H_i(\mathbf{X}^i)$ megoldást. A P_i ezek után egy C_i lépést hajt végre az $\mathbf{y}_k$ megoldásból kiindulva, vagyis $\mathbf{x}_{k+1}^i \in C_i(\mathbf{y}_k)$. Tehát ezzel lényegében az $\mathbf{x}_{k+1}^i \in C_i H_i(\mathbf{X}^i, \mathbf{x}_k^i)$ megoldást kaptuk.

6. Tétel. *(A globális dekompozíció aszinkron konvergenciája)*
Tegyük fel, hogy a $C_i, i = 1, \ldots, N$ algoritmusok konvergensek, a H_i algoritmusok leszálló algoritmusok az $\mathbf{X}^i$ és a g szerint, valamint tegyük fel, hogy a teljes aszinkronitás TA feltétele fennáll, akkor
(i) a $\{\mathbf{x}^i(t)\}, t \in T^i$ megoldás sorozat tetszőleges konvergens részsorozata minden $i = 1, 2, \ldots, N$ esetén konvergál egy optimális megoldáshoz, továbbá
(ii) ha a P^i processzornál a $\underline{t}$ időpontban rendelkezésünkre áll egy $\mathbf{x}^i(\underline{t})$ megoldás, akkor van olyan $\bar{t} > \underline{t}$ időpillanat, hogy $g(\mathbf{x}^j(t)) \leq g(\mathbf{x}^i(\underline{t}))$, ha $t \geq \bar{t}$ minden $j = 1, 2, \ldots, N$ index esetén.

Bizonyítás. A tétel (i) része egyszerű következménye a 5. Tételnek, ha azt minden egyes processzorra alkalmazzuk. A H_i pótlólagos algoritmikus lépés (a P_i processzornál a C_i algoritmikus lépés után beillesztett heurisztika, mondjuk a (2.4)-ben adott minimalizálás) úgy tekinthető, mint a távolságtartó lépések közé beillesztett közbülső lépés. Az $\{\mathbf{x}^i(t)\}, t \in T^i$ megoldások halmaza megfelel az 5. Tételben megadott $\{\mathbf{x}_k\}, k \in \mathcal{K}$ halmaznak, amely konvergens, az $\{\mathbf{y}_k^i\}$ vektorok halmaza pedig a $\{\mathbf{x}_k\}, k \notin \mathcal{K}$ halmaznak.

A tétel (ii) részének bizonyításához tekintsük azt a $\underline{t} \in T^i$ pillanatot, amikor P_i befejezte a $\mathbf{x}^i(\underline{t})$ kiszámítását. A TA teljes aszinkronitás feltevésének második része miatt $\lim_{t\to\infty} \tau_i^j(t) = \infty$, azaz létezik egy olyan $\underline{t}_j$ időpillanat, amelyre $\tau_i^j(t) > \underline{t}$ fennáll minden $t > \underline{t}_j$ esetén. Ebből pedig az következik, hogy $\tau_i^j(\underline{t}_j)$ időpillanatra $\mathbf{x}^i(\underline{t})) \in \{\mathbf{X}^j\}_i$ igaz, továbbá $g\left(\mathbf{x}^i(\tau_i^j(t))\right) \leq g(\mathbf{x}^i(\underline{t})),\ \forall t \geq \underline{t}_j$.

Legyen t_j^* az az idő, amelyre P_j-nek szüksége van ahhoz, hogy a jelenlegi H_i iterációt befejezi, majd a C_i lépést elvégzi, valamint az ehhez kapcsolódó H_i leszálló lépést (itt t_j^* véges a TA feltevés első része miatt), amivel kiszámítja a $\mathbf{x}^j(\underline{t}_j + t_j^*)$ értéket.

Mivel a $\underline{t}_j$ időre a $\mathbf{X}^j$ tömb tartalmazza már a $\mathbf{x}^i(\underline{t})$ értéket, így $g(\mathbf{x}^j(\underline{t}_j + t_j^*)) \leq g(\mathbf{x}^i(\underline{t})$ fennáll. Ha most $\overline{t}$-vel jelöljük az összes ilyen idő maximumát: $\overline{t} = \max_{j=1,2,\ldots,N}(\overline{t}_j + t_j^*)$, akkor tudjuk, hogy a $\overline{t}$ időre az összes $P_j, j = 1, \ldots, N$ processzoroknak olyan megoldások állnak a rendelkezésükre, amelyeken a felvett függvényértékek legalább olyan kicsik, mint a P_i processzornál voltak a $\underline{t}$ időben.□

A tétel (ii) része azt mutatja, amit a legjobb megoldás elterjedésének nevezhetünk; ha egyszer egy processzor már megtalált egy jó közelítést, akkor némi idő után az összes processzornál legalább ilyen jó függvényértéket adó közelítések lesznek. A H_i algoritmusok által gyakran használt $g(\mathbf{x}^j(t))$ függvényértéket az egyes processzorok szétküldik azért, hogy a többi processzor ne végezzen felesleges munkát. A k iterációs számláló értékét azért küldik el az $\mathbf{x}^j(t)$ megoldással együtt, hogy egy másik processzor mindig a legutolsó kiszámított (és nem a legutolsó megkapott) megoldást tárolja. Ez az óvintézkedés annak a nehézségnek az elkerülésére szolgál, hogyha némely, processzorok közti üzenet késik, vagyis ha egy korábban P_j-ből P_i-be küldött megoldás később ér a rendeltetési helyére, mint egy később kiszámított és elküldött megoldás, akkor ne cseréljünk. Matematikailag ez feleslegessé teszi a $\tau_j^i(t)$ függvény monotonitásának megkövetelését.

2.4. Megfontolások az általános modellről

A közölt általános dekompozíciós modell fő előnye az, hogy a 6. Tétel szerint ez automatikusan kiválasztja a rendelkezésre álló legjobb megoldást minden iteráció után. Így az átlagosnál jobb konvergenciasebesség elvárható a következő megfontolások alapján. Egy optimalizálandó függvény más és más pontokban (ezeknek egy kis környezetében) lényegesen eltérő módon viselkedhet (kivéve természetesen azt az esetet, ha valamilyen globális jó tulajdonságot, például konvexséget, felteszünk a teljes függvényről). Ha különböző optimalizálási algoritmusokat rendelünk a különböző processzorokhoz és az egyik algoritmus felfedez – a függvény egy lokális tulajdonsága és az algoritmus egy speciális vonása miatt – egy nagy csökkenést a függvényértékben, akkor némi idő elteltével a többi processzor is képes ezt a nagy ugrást követni, és ebben az új pontban (vagy ennek egy környezetében) egy másik algoritmus lehet sikeres.

A fentieknek, pontosabban a 6. Tétel (ii) részének a jelentőségét a következő példával szemléltetjük. Közismert, hogy a Newton módszer és különböző változatai kvadratikusan

konvergálnak, ha az aktuális megengedett megoldás már „elég közel" van az optimumhoz. Szintén köztudott, hogy az esetek többségében a véletlen kereső eljárások jobban működnek, ha „messze" vagyunk az optimális megoldástól. Itt az „elég közel" és a „messze" kifejezések definiálatlan és intuitíve leírt fogalmak; egy tényleges optimalizálási feladat numerikus végrehajtása során nagyon nehéz, vagy lehetetlen kijelenteni, hogy „most már elég közel vagyunk az optimumhoz, és ezért a véletlen keresésről váltsunk át a Newton módszerre". A közölt globális dekompozíciós eljárás éppen ezt a döntést hozza meg helyettünk, mivel automatikusan a jobban működő eljárást választja ki.

A globális dekompozíciós modell még a párhuzamos számítógépben (összekapcsolt munkaállomások hálózatában) fellépő hibákat is tudja kezelni. Ha néhány processzor közötti kapcsolat lelassul, vagy megszakad, vagy néhány processzor teljesen leáll, a globális dekompozíció még mindig működik és képes az optimális megoldást megadni – mindaddig, amíg egyetlen olyan processzor működik, amely konvergens algoritmussal is rendelkezik. Ugyanis nyilvánvalóan elvégezhetjük az algoritmusoknak a processzorokhoz való rendelését olyan módon, hogy egyes processzorok csak leszálló H_i algoritmus lépéseit végzik, de a többi processzor a távolságtartó lépések tételének értelmében még mindig egy konvergens sorozatot képes előállítani.

Az előző szakaszban közölt általános számítási modell nagyon alkalmas olyan sztochasztikus programozási feladat [PGDP 76] párhuzamos számítógépen való elvégzésére, amelyben van egy „nehezen kiszámítható" függvény. Például ha a normális eloszlásfüggvény értékeit Monte Carlo becsléssel közelítjük, ezt a munkát több processzor egyidejűleg végezheti, és az eredményűl kapott, véletlen hibával terhelt értékeket egy (vagy több) kitüntetett szerepet játszó processzorhoz küldik, amely a kapott értékekből az eloszlásfüggvény közelítésére egy regressziós felületet tud előállítani [De 98b]. De teljesen hasonló helyzet áll elő a sztochasztikus programozás kétlépcsős feladata során: itt a várható értékként definiált várható pótlás függvény értékeinek szimulálására, illetőleg regressziós görbével való közelítésére van szükség [De 03b].

Hasonlítsuk most össze a globális dekompozíciót a komponensenkénti dekompozícióval. A közölt eljárásban nincs szükség a BC doboz előírás feltételezésére, vagyis a globális eljárás általános optimalizálási eljárások esetén is használható, nem csak dekomponálható struktúrájú fixpont feladatokra. Viszont ez az általános módszer működik a teljes aszinkronitás TA feltétele mellett is. Továbbá a számítógépen fellépő hibák kezelése is jóval hatékonyabb, hiszen a komponensenkénti dekompozíció esetén elég egy olyan processzor (vagy az ettől érkező adatok továbbítására szolgáló hálózati rész) kiesése, amely a megoldás egy komponensét számítja, és az egész optimalizálás leállhat.

Az eljárás hátrányai közül elsőként említjük, hogy a komponensenkénti dekompozícióhoz hasonlóan, itt is csak globális konvergenciát tudunk garantálni; a felhasznált N processzor számával arányos gyorsulást nem tudjuk biztosítani. A legrosszabb esetben még az is megtörténhet, hogy a konvergencia csak olyan erős lesz, mint amilyenre a felhasznált legjobb C_i algoritmus lenne képes egy szekvenciális számítógépen.

A globális dekompozíció egy további hátránya, hogy minden egyes processzornak a

teljes feladathoz hozzáférést kell biztosítani (vagy a processzor lokális memóriájában, vagy egy gyors hálózaton át), valamint a teljes C_i, H_i algoritmusokat is lokálisan be kell tölteni. Ez a rendelkezésre álló memóriát megterhelheti.

2.5. A globális dekompozíció változatai

A 6. Tételben megadott konvergenciára vonatkozó eredményből látható, hogy a legrosszabb esetben a teljes párhuzamos számítógépen csak a legjobb C_i algoritmusnak megfelelő (szekvenciális) konvergencia sebesség érhető el. Itt a H_i algoritmusoknak néhány olyan változatát írjuk le, amelyek remélhetőleg valamivel jobb konvergenciasebességet biztosítanak. Feltesszük a használható algoritmusok nagy részének ismerete alapján, hogy a C_i konvergens algoritmusok egy lépésének végrehajtásához nagyon sok számításra van szükség, míg a H_i algoritmikus lépéshez viszonylag kevés munka kell, hiszen ennek csak leszálló algoritmusnak kell lennie. Vagyis a H_i algoritmusok használatának az alapötlete az, hogy viszonylag nagy javulás érhető el a célfüggvény (a leszálló függvény) értékében, viszonylag kis munkával.

2.5.1. A konvergens algoritmusok kiválasztása

Néhány szempontot adunk meg a C_i konvergens algoritmusok kiválasztásának mérlegeléséhez. Ezek a $C_i, i = 1, \ldots, N$ algoritmusok vagy egymástól teljesen különbözőek lehetnek, vagy pedig ugyanannak az eljárásnak a különböző paraméterekkel, vagy kezdőponttal ellátott másolatai. Azt javasoljuk, hogy az algoritmusok halmazának a rendelkezésünkre álló optimalizálási eljárások széles körét ölelje fel:

- egy véletlen kereső algoritmus, lehetőleg különböző, a megengedett megoldások tartományát lehetőleg egyenletesen lefedő kezdeti értékekkel és különböző paraméterekkel – lépéshossz, hibahatárok, stb. – hogy elég jó konvergenciát kapjunk az optimumtól „messze”,
- a Newton módszer, vagy annak egy variánsa, hogy ha „elég közel” vagyunk az optimumhoz, akkor kvadratikus konvergenciát kapjunk,
- a részleges konjugált irányok módszerét, hogy megszabaduljunk a célfüggvény Hesse mátrixában esetleg meglévő lényegesen különböző sajátértékek által okozott numerikus nehézségektől,
- a legmélyebb leszállás – gradiens módszer, hogy egy megbízható, majdnem mindig megfelelően működő általános konvergenciát adó módszerünk legyen,
- a feladattól és lehetőségeinktől függően SUMT módszerek, metszősík módszerek, belső pontos módszerek, általánosított Lagrange eljárások, stb.

Előnyösnek tűnik az a megoldás, hogy ha ugyanazt a C_i algoritmust lényegesen különböző kezdeti pontokból kiindulva használjuk, hiszen így a különböző pontokban lévő lokális viselkedést ki tudjuk használni – lásd fentebb az első pontnál. Egy ilyen eljárást írtunk le metszősík algoritmusokra a [De 89] cikkben.

2.5.2. A heurisztikus algoritmusok kiválasztása

Egy használható H_i leszálló algoritmust két különböző részből kell előállítani: egy kooperációs lépésnek nevezett D_i részből, amely az $\mathbf{X}^i$-ben tárolt összes megoldást felhasználja, valamint egy E_i perturbációs részből, amely a degeneráció elleni óvintézkedést jelenti. A degeneráció itt mindössze azt jelenti, hogy két processzor (algoritmus) ugyanazt a módszert használva, majdnem ugyanazon pontból kiindulva, lényegében ugyanazt a munkát végzi, feleslegesen. A legtöbb esetben az E_i algoritmus lehet üres is, semmilyen munkát nem kell végezni. Ebben a szakaszban az $\mathbf{X}^i$ tömbben tárolt $\mathbf{x}^j(\tau_j^i(t))$ megoldásokat egyszerűen a $\mathbf{x}^j, j = 1, \ldots, N$ jelöléssel fogjuk használni.

A kooperációs lépés lényegében csak egy D_i transzformáció, amely segítségével az ismert $\mathbf{x}^j, j = 1, \ldots, N$ megoldásokból egy olyan $\mathbf{y}$ megoldást akarunk előállítani, amelyen a célfüggvény értéke nem nagyobb, mint az $\mathbf{x}^j, j = 1, \ldots, N$ akármelyikén, vagyis $g(\mathbf{y}) \leq g(\mathbf{x}^j)$, ha $\mathbf{y} \in D_i(\mathbf{x}^j)$, $\forall j \in J$. Ebben a részben néhány lehetőséget írunk le a D_i algoritmusokra, míg a következő részben az E_i perturbációs algoritmusokat vizsgáljuk.

A megoldások minimuma

Ezt a lehetőséget a (2.4) egyenletben említettük, vagyis ennek alapján meghatározzuk a $\mathbf{y}^{i,min} = \text{argmin}_{j=1,\ldots,N}\ g(\mathbf{x}^j)$ értéket. Ezt az eljárást csak akkor ajánljuk, ha a C_i algoritmusok mind különbözők (vagy ha lényegesen különböző paraméterekkel futnak). Egyébként a D_i algoritmus ilyen választása vagy ugyanazt a pontot, vagy egészen közeli pontsorozatot állíthat elő – következésképpen nagyon gyengén használjuk ki a párhuzamosságot. Az $\mathbf{y}^{i,min}$ előállítását viszonyítási pontként használhatjuk a továbbiakban, ennél jobb eredményeket szeretnénk elérni.

Minimalizálás egy egyenes mentén

Tekintsük az $\mathbf{x}^i + \lambda(\mathbf{x}^j - \mathbf{x}^i),\ \lambda \geq 0$ félegyenest és minimalizáljuk a $g(\cdot)$ függvényt ennek mentén. Legyen $\mathbf{x}^{ij}$ az a pont, amelyen a minimum felvétetik, azaz

$$\mathbf{y}^{ij} = \text{argmin}_\lambda\ g(\mathbf{x}^i + \lambda(\mathbf{x}^j - \mathbf{x}^i)) \tag{2.5}$$

Jegyezzük meg, hogy ennek a minimalizálásnak nem kell pontosnak lennie, elég egy olyan $\mathbf{z}$ pontot találni, amelyen a függvényérték kisebb, mint az eddigi megoldásokon elért függvényérték, azaz $g(\mathbf{z}^{ij}) \leq g(\mathbf{x}^j), j = 1, \ldots, N$ elfogadható. Amint ezt a minimalizálást elvégeztük az $\mathbf{x}^i$ pontot és az $\mathbf{x}^j, j = 1, \ldots, N, j \neq i$ pontokat összekötő egyeneseken, válasszuk a legjobb pontot.

Egy másik lehetséges D_i változat lehet, ha a minimalizálást az $\mathbf{x}^i$ pontból kiinduló, az $\mathbf{y} = \text{argmin}_{j \in J}\, g(\mathbf{x}^j),\ J = \{j | j = 1, 2, \ldots, N, j \neq i\}$ ponton átmenő félegyenesen végezzük el, miáltal ez a lépés egy kvázi-gradienses lépés lesz. Mivel az $\mathbf{y}$ kiválasztása egyszerű és csak egyetlen vonal menti minimalizálásra van szükség, ezt az algoritmust ajánljuk.

Szimplex technika

A terjeszkedő és összezsugorodó szimplexek Nelder és Mead, valamint Kowalik és Osborne [BSS 94] által javasolt eljárása is használható. Előnye, hogy nincs szükség egyenes menti minimalizálásra, csak olyan függvényértékek összehasonlítására, amelyek már úgyis rendelkezésünkre állnak az $\mathbf{X}^i$ tömbben. Egy kezdeti szimplex kialakításához $n+1$ megoldásra van szükségünk (ahol n a primál megengedett $\mathbf{x}$ megoldás dimenziója).

Az n dimenziótól és a processzorok N számától függően két esetet kell itt megkülönböztetni. A feladatot túldimenzionáltnak nevezzük, ha több processzor van, mint a megoldásvektor dimenziója, és aluldimenzionáltnak, ha $N \leq n$. Az első esetben kiválaszthatjuk az $n+1$ legjobb megoldást a rendelkezésünkre álló $\{\mathbf{x}^j,\ j \in J, \mathbf{x}^i\}$ vektorokból.

Az aluldimenzionált esetben néhány további $\mathbf{y}_l, l = 1, 2, \ldots, N-n$ pontot kell előállítanunk. Ez elvégezhető például a $\overline{\mathbf{x}} = \frac{1}{N}\sum_{j=1}^{N} \mathbf{x}^j$ súlypont véletlen perturbálásával, véletlen eltolások generálásával. Legyen az $\mathbf{x}^j$ pontok átlagos távolsága a $\overline{\mathbf{x}}$ ponttól d, vagyis

$$d^2 = \frac{1}{N}\sum_{j=1}^{N}\sum_{m=1}^{n}(x_m^j - \overline{x}_m)^2, \tag{2.6}$$

és generáljuk a $\{\boldsymbol{\xi}_s\}$ véletlen vektorokat egyenletesen az $U = \{\mathbf{x} | \sum_i x_i^2 = 1\}$ egységgömb felületén. Ekkor az

$$\mathbf{y}_s = \overline{\mathbf{x}} + d\boldsymbol{\xi}_s, s = 1, 2, \ldots, N-n \tag{2.7}$$

pontokat használhatjuk pótlólagos pontokként a kezdeti szimplex hiányzó csúcsainak szerepében. Vegyük észre, hogy a szimplex algoritmust itt nem optimalizálásra használjuk, ezért ennek végrehajtásában csak két–három lépés kiszámítását (két–három egymásutáni polihedron meghatározását) ajánljuk. Mivel nem hajtjuk végre a szimplex módszert teljes egészében, ezért használhatunk a szimplex helyett egy elfajuló, $n+1$-nél kevesebb csúccsal rendelkező polihedront – vagyis teljeséggel megengedett csak a rendelkezésre álló $\mathbf{x}^j$ megoldásokat használni.

2.5.3. Perturbációs lépés

A perturbációs lépés használatára akkor van szükség, ha a D_i transzformáció eredményeként egy olyan pontot kapunk, amely már rendelkezésünkre áll az $\mathbf{X}^i$ tömbben, vagyis ha $\mathbf{y} \in D_i(\mathbf{x}^1, \ldots, \mathbf{x}^N)$ egyenlő (vagy nagyon közel van) az $\mathbf{x}^j$ ponttal valamilyen j index esetén. Másszóval a perturbáció segítségével N különböző pontot tartunk fenn az összes processzornál. Az E_i perturbációs lépés segítségével egy olyan $\mathbf{z}$ pontot kell előállítani,

amely különbözik az összes rendelkezésünkre álló $\{\mathbf{x}^j, j \in J\}$ megoldástól, de egyébként a megengedett megoldások halmazában marad. Itt olyan pontokat is elfogadhatunk, amelyek esetén a függvényérték növekszik, de azért a konvergens algoritmusok által előállított legjobb megoldáson elért függvényérték alatt marad, tehát a $g(\mathbf{z}) \leq g(\mathbf{y}^{i,min})$ egyenlőtlenség teljesülését megköveteljük.

Egy lehetőség perturbált megoldások előállítására az lehet, hogy egy kicsit „elrontjuk" a már valamilyen minimalizálási eljárással előállított legjobb megoldást, hogy egy kicsit rosszabb, de különböző megoldást kapjunk. A kezdeti szimplex előállításának esetén követett eljáráshoz hasonlóan járhatunk el: addig generálunk véletlen pontokat az egységgömb felületén és adjuk hozzá a súlyponthoz, amíg egy olyan $\mathbf{y}_s = \overline{\mathbf{x}} + d\boldsymbol{\xi}_s, s = 1, 2, \ldots$ véletlen pontot nem kapunk, amelyre $g(\mathbf{y}_s) \leq g(\mathbf{y}^{i,min})$ fennáll. Természetesen ezt az eljárást követhetjük az egységgömbben lévő $\boldsymbol{\xi}$ pontok esetén is.

A perturbálásra egy további lehetőség az, ha a minimalizálási lépésben kapott pontos minimum értékeként kapott pontot addig perturbáljuk, amíg egy megfelelő eredményt nem kapunk.

Végül felhívjuk a figyelmet arra, hogy a szimulált hűtés eljárását is alkalmazhatjuk egy leszálló algoritmusként, de itt is külön ellenőrizni kell a kapott ponton a függvény csökkenését. □

3. fejezet

Konvex halmazok valószínűségének kiszámítása normális eloszlás esetén

Ebben a fejezetben néhány egyszerű konvex halmaz valószínűségének Monte Carlo meghatározásával foglalkozunk normális eloszlás esetén. Megadjuk az általános poliéder, ellipszoid és körkúp esetére vonatkozó részletes algoritmusokat is. Ezek segítségével néhány nemkonvex eset kezelése is lehetővé válik.

3.1. Bevezetés

A feladatot általános formában a következőképpen fogalmazhatjuk meg. Legyen a 0 várhatóértékű, R korrelációmátrixszú, nem elfajult normális eloszlás sűrűség-, illetőleg eloszlásfüggvénye φ és Φ, ahol R pozitív definit. Ekkor

$$\varphi(\mathbf{z}) = (2\pi)^{(-n/2)}|R|^{-\frac{1}{2}} \exp\left\{-\frac{1}{2}\mathbf{z}'R^{-1}\mathbf{z}\right\}, \tag{3.1}$$

$$\Phi(\mathbf{h}) = \int_{-\infty}^{h_1} \cdots \int_{-\infty}^{h_n} \varphi(\mathbf{z})d\mathbf{z}, \tag{3.2}$$

más, nem-standard normális eloszlás esete lineáris transzformációval kezelhető [Ton 90], [KBJ 00]. Az n-dimenziós X halmaz valószínűségét ekkor a következő integrál adja meg:

$$Pr\{X\} = \int_X \varphi(\mathbf{z})d\mathbf{z}. \tag{3.3}$$

Ilyen integrálok kiszámítására van szükségünk sztochasztikus programozási feladatok megoldása [KM 96], [May 92], [Pr 95], [PGDP 76], szerkezetek megbízhatóságával kapcsolatos feladatok [Bje 90], [DB 89], vagy többdimenziós statisztikai feladatok esetén [Bre 94], [DC 86]. A $Pr\{X\}$ meghatározása még egyszerű halmazok esetén is numerikusan nehéz feladat, különösen igaz ez magasabb dimenzió esetén, ha $n \geq 10$. A dimenziós

robbanás miatt ilyen esetekben csak Monte Carlo módszerekkel lehet hatékony eljárást megadni a valószínűség kiszámítására [De 88].

Három Monte Carlo módszer ismeretes az n-dimenziós eloszlásfüggvény értékeinek kiszámítására, vagyis a $\Phi(\mathbf{h}) = Pr\{H\}, H = \{\mathbf{x} \mid \mathbf{x} \leq \mathbf{h}\}$ érték statisztikai becslésére. A szerző által javasolt ortonormált becslések módszere (amely eredeti formájában a [De 79], [De 80a], [De 90a] publikációkban található, ennek továbbfejlesztése téglatestek esetére a [De 86b] cikkben). A második egy Szántai által javasolt szita formulát használó eljárás [Sz 86], ennek Prékopa által továbbfejlesztett változata, egy Boole–Bonferroni egyenlőtlenségeken alapuló eljárás [Pr 95], valamint ehhez kapcsolódóan Szántai és Bukszár hipercseresznyefákat használó eljárása [BP 01], [BS 02]. Végül a harmadik algoritmus a két előző módszer kombinálásával kialakított Gassmann-féle hibrid eljárás. Ezek összehasonlítása a [Ga 88] cikkben, a legújabb eredmények numerikus kiértékelése pedig a [GDS 02] cikkben található. A felhasználhatóság szempontjából megemlítjük, hogy Prékopa, Bukszár és Szántai eredményei tetszőleges eloszlásfüggvény kiszámítására használhatók, míg az itt közölt eljárások csak normális eloszlás esetén alkalmazhatók, de többféle konvex halmazra.

Ebben a fejezetben az eloszlásfüggvényre alkalmazott ortonormált becslések módszerét kiterjesztjük általános konvex poliéderek, hiperellipszoidok és körkúpok esetére. Megjegyezzük, hogy az eljárást Lohr alkalmazta csillag alakú halmazokra [Loh 93] és Monahan alkalmazta a t-eloszlásra [MG 97].

Az algoritmusokat FORTRAN nyelvű programokban valósítottuk meg, ezek a szubrutinok alkotják a NORSET programcsomagot. A számítógépes kisérletezések szerint $n \leq 20$ dimenziókban három decimális jegy pontosságú becsléseket lehet kapni egy másodpercnél rövidebb idő alatt még a legrosszabb esetekben is, ami az általánosan használható elfogadás-elvetés eljárással összehasonlítva mintegy 20-100-szoros gyorsulást mutat. A $20 \leq n \leq 100$ dimenziós esetekben ugyan kisebb gyorsulást tapasztaltunk, de még mindig elfogadható idő (kevesebb mint 5 másodperc) alatt kapható gyakorlatilag használható, három tizedesre pontos eredmény.

A következő részben az ortonormált becslések általános Monte Carlo módszerét írjuk le a $\Phi(\mathbf{h})$ eloszlásfüggvény kiszámítása esetén, majd a megfelelő képleteket vezetjük le poliéder, hiperellipszoid és körkúp valószínűségének meghatározásához. A negyedik részben megjegyzéseket teszünk arra vonatkozóan, hogy általános halmazok valószínűségének kiszámítása hogyan végezhető, az utolsó szakaszban pedig a számítógépes kisérletezések eredményeiből adunk egy rövid áttekintést a [De 03a] cikk alapján (további számítógépes futások adatait tartalmazzák a [De 98c], [De 00] cikkek).

3.2. Iránymenti integrálás – ortonormált becslések

Tekintsük feladatunknak az

$$I = Pr\{X\} = \int_X \varphi(\mathbf{z})d\mathbf{z} = \int_{R^n} f(\mathbf{z})d\Phi(\mathbf{z}) \tag{3.4}$$

integrál kiszámítását, ahol $f(\mathbf{z})$ az X halmaz indikátorfüggvénye, azaz

$$f(\mathbf{z}) = \begin{cases} 1, & \text{ha } \mathbf{z} \in X, \\ 0, & \text{egyébként.} \end{cases}$$

A (3.4) egyenlőség jobboldala alapján a következő Monte Carlo eljárást lehet megadni a $Pr\{X\}$ valószínűség kiszámítására. Legyen a $\boldsymbol{\xi}$ valószínűségi változó φ sűrűségfüggvényű, és jelölje ennek a független mintáit $\mathbf{x}_i, i = 1, \ldots, N$. Ekkor I egy torzítatlan becslése a

$$\Theta_1 = \frac{1}{N}\sum_{i=1}^{N} f(\mathbf{x}_i) \tag{3.5}$$

átlag. Ezt a becslést a durva becslésnek, vagy elfogadás-elvetés becslésnek nevezzük, mivel a gyakorlatban ez egy olyan eljárásra vezet, ahol a $\mathbf{x}_i, i = 1, \ldots, N$ vektorok generálása után már csak az X halmazban fekvő vektorok számát kell meghatározni, így a becslés a relatív gyakoriság.

Közismert, hogy egy normális eloszlású $\boldsymbol{\xi}$ vektor felírható

$$\boldsymbol{\xi} = \chi_n T\boldsymbol{\eta} \tag{3.6}$$

alakban, ahol χ_n egy n szabadságfokú χ-eloszlású skalár valószínűségi változó, T egy felső–háromszög alakú mátrix, amelyre $TT' = R$ fennáll, továbbá az $\boldsymbol{\eta}$ valószínűségi vektorváltozó egyenletes eloszlású az egységgömb felületén, az $S = \{\mathbf{x} \mid \sum_{i=1}^{n} x_i^2 = 1\}$ halmazon. A $\boldsymbol{\xi}$ vektor χ_n „hossza” és az $\boldsymbol{\eta}$ „iránya” független valószínűségi változók.

Jelölje a χ-eloszlású χ_n valószínűségi változó eloszlásfüggvényét $K(\lambda), \lambda \geq 0$ és az $\boldsymbol{\eta}$ valószínűségi vektorváltozó eloszlásfüggvényét $V(\mathbf{y}), \mathbf{y} \in S$. Ezekkel a jelölésekkel az integrálunk átírható:

$$I = \int_{R^n} f(\lambda T\mathbf{y})dK(\lambda)dV(\mathbf{y}) = \int_S \left(\int_0^\infty f(\lambda T\mathbf{y})dK(\lambda) \right) dV(\mathbf{y}). \tag{3.7}$$

Vezessük be a $g(\mathbf{y})$ jelölést a belső integrálra, legyen

$$g(\mathbf{y}) = \int_0^\infty f(\lambda T\mathbf{y})dK(\lambda). \tag{3.8}$$

Ez a $g(\mathbf{y})$ függvény egy rögzített $\mathbf{y}$ esetén megadja a $\lambda T\mathbf{y}, \lambda \geq 0$ sugár X halmazba eső részének a valószínűségi tartalmát. Tegyük fel most, hogy a $\lambda\mathbf{z} = \lambda T\mathbf{y}$ egyenes elmetszi

az X konvex halmazt, és definiáljuk az egyenes belépési, illetőleg kilépési pontjait a λ_L és λ_U konstansok segítségével (ahol $X \cap \{\mathbf{z} \mid \mathbf{z} = \lambda T\mathbf{y}\} = [\lambda_L T\mathbf{y}, \lambda_U T\mathbf{y}]$), azaz

$$\lambda_L = \min_\lambda \{\lambda \mid f(\lambda T\mathbf{y}) = 1\} = \min_\lambda \{\lambda \mid \lambda T\mathbf{y} \in X\}, \tag{3.9}$$
$$\lambda_U = \max_\lambda \{\lambda \mid f(\lambda T\mathbf{y}) = 1\} = \max_\lambda \{\lambda \mid \lambda T\mathbf{y} \in X\}.$$

Mivel ezek a λ_L, λ_U értékek függenek az $\mathbf{y}$ vektortól, ezért a $K(\lambda_L) = K(\lambda_L \mid \mathbf{y}), K(\lambda_U) = K(\lambda_U \mid \mathbf{y})$ jelölésekkel is fogunk élni. (Jegyezzük meg, hogy az egydimenziós $K(\cdot)$ eloszlásfüggvény könnyen kiszámítható létező szoftverek segítségével.) Bevezetve a $\lambda_L^+ = \max\{0, \lambda_L\}$, $\lambda_U^+ = \max\{0, \lambda_U\}$ jelöléseket, látható, hogy a $g(\cdot)$ függvény a

$$g(\mathbf{y}) = K(\lambda_U^+|\mathbf{y}) - K(\lambda_L^+|\mathbf{y}). \tag{3.10}$$

formába írható. Hasonlóan kapható a $\lambda_U^- = \min\{0, \lambda_U\}, \lambda_L^- = \min\{0, \lambda_L\}$ jelölések használatával

$$g(-\mathbf{y}) = -K(-\lambda_U^-| - \mathbf{y}) + K(-\lambda_L^-| - \mathbf{y}),$$

vagyis a λ_L, λ_U állandók meghatározása után a $\lambda\mathbf{z} = \lambda T\mathbf{y}$ egyenes valószínűségi tartalma is meghatározható:

$$e(\mathbf{z}) = e(T\mathbf{y}) = [g(\mathbf{y}) + g(-\mathbf{y})]/2.$$

A belső integrálra kapott (3.10) kifejezést visszahelyettesítve a (3.7) kettős integrálba írhatjuk, hogy

$$I = \int_S \left[K(\lambda_U^+|\mathbf{y}) - K(\lambda_L^+|\mathbf{y})\right] dV(\mathbf{y}). \tag{3.11}$$

Az egyenlet alapján a következő Monte Carlo eljárás adható az I kiszámítására: generáljuk az $\mathbf{y}_i, i = 1, \ldots, N$ független mintákat az S-en adott egyenletes eloszlásból és számítsuk ki determinisztikusan (a megfelelő program segítségével) a g értékét minden $\mathbf{y}_i$ esetén. Tehát becslésünk I-re a következő:

$$\Theta_2 = \frac{1}{N}\sum_{i=1}^{N} g(\mathbf{y}_i) = \frac{1}{N}\sum_{i=1}^{N} K(\lambda_U^+|\mathbf{y}_i) - K(\lambda_L^+|\mathbf{y}_i). \tag{3.12}$$

Ez a becslés az I-t a következő módon határozza meg: a (3.8) belső integrált determinisztikusan számítjuk, míg a külső integrálra mintavételt hajtunk végre. A becslést hatékonyabbá tudjuk tenni a következő két módosítás segítségével.

Az egyik módosítást arra alapozzuk, hogy a Θ_2 szórása viszonylag nagy, mert az $\mathbf{y}_i$ vektorok „túl véletlenszerűen" vannak szétszórva az egységgömb felületén, így egy olyan elrendezésre van szükségünk, amely a felhasznált vektorok „egyenletességét" növeli. Ez a következő módon érhető el: a független $\mathbf{y}_i$ vektorok helyett egy ortonormalizált vektorokból álló rendszert állítunk elő.

Tekintsük a véletlen U rendszert, amely S-ben lévő, ortonormalizált vektorokból áll; legyen $U = \{\mathbf{u}^i, i = 1, \ldots, n \mid \mathbf{u}^i \in S, \mathbf{u}^{i'}\mathbf{u}^j = \delta_{ij}, i, j = 1, \ldots, n\}$, ahol $\delta_{ij} = 0, i \neq j, \delta_{ii} =$

1, és U egyenletes eloszlású az ortonormalizált rendszerek felett. A második módosítás előkészítéseként tekintsük tetszőleges két U-beli vektor összegét és különbségét, vagyis legyen

$$\mathbf{v}^{i,j,\mathbf{s}} = \frac{1}{\sqrt{2}} \left(s_1 \mathbf{u}^i + s_2 \mathbf{u}^j \right), \tag{3.13}$$

ahol az i, j indexpár és az $\mathbf{s}$ előjelvektor az összes lehetséges értéket felveszi a

$$J^* = \{(i, j, \mathbf{s}) \mid i = 1, \ldots, n-1, j = 2, \ldots, n, i < j, s_1 = 1, s_2 = 1, \text{vagy } s_1 = 1, s_2 = -1\}$$

halmazból. Az $\mathbf{u}^i$ vektorok normalizált összegét azért használjuk, hogy az egy vektorra eső számítási munkát csökkentsük. Egy adott U rendszer esetén csak n számú $\mathbf{u}^i$ vektorunk van, de az U-ból előállítható $\mathbf{v}^{i,j,\mathbf{s}}$ vektorok száma $2n(n-1)$ (az előállított egyenesek száma pedig $n(n-1)$).

A másik módosítást a következő, számítástechnikai szempontból fontos megjegyzésre alapozzuk. A $T\mathbf{v}^{i,j,\mathbf{s}}$ transzformált vektorok kiszámítása helyett csak az $\mathbf{u}^i$ vektorok $T\mathbf{u}^i$ transzformált vektorait határozzuk meg, mivel ezek segítségével az előbbiek megadhatók, ugyanis

$$\mathbf{z}^{i,j,\mathbf{s}} = T\mathbf{v}^{i,j,\mathbf{s}} = \frac{1}{\sqrt{2}} \left(s_1 T\mathbf{u}^i + s_2 T\mathbf{u}^j \right),$$

tehát $n(n-1)$ mátrixszorzás helyett csak n mátrixszorzásra lesz szükségünk a $T\mathbf{v}^{i,j,\mathbf{s}}$ vektorok előállításához. Így az egy generált vektorra eső számítási munka lényegesen csökkenthető, továbbá a generált vektorok „egyenletessége" is növelhető.

A két módosítás figyelembevételével megadjuk egy teljes U rendszer esetén a becslés formális alakját akkor, ha tetszőleges két U-beli vektor összegére és különbségére végezzük el a függvényértékek kiszámítását:

$$O_2 = \frac{1}{n(n-1)} \sum_{(i,j,\mathbf{s}) \in J^*} e\left(T\mathbf{v}^{i,j,\mathbf{s}}\right) = \frac{1}{n(n-1)} \sum_{(i,j,\mathbf{s}) \in J^*} e\left(\frac{1}{\sqrt{2}} \left(s_1 T\mathbf{u}^i + s_2 T\mathbf{u}^j \right) \right). \tag{3.14}$$

Ezt a becslést ortonormalizált–2 (ortonormált–2), vagy röviden O_2 becslésnek nevezzük. Természetesen a gyakorlati számítások során N különböző U rendszert generálunk véletlenszerűen, és az eredményül kapott valószínűségek átlagát számítjuk ki.

Két darab, U rendszerből származó vektor összege és különbsége helyett vehetnénk k darab U-beli vektor minden lehetséges előjellel vett normalizált összegét is, ezáltal az O_k becslést kapnánk – ezzel megnövelnénk az egy rendszerből előállítható vektorok számát és csökkentenénk az egy előállított vektorra eső számítási munkát. Az így kapott O_k becslések formális leírása könnyen megkapható a fentiek alapján, így ezek leírását mellőzzük. Az O_2 becslések általában számítástechnikailag megfelelőnek bizonyultak (az O_k becslések összehasonlítására vonatkozó numerikus eredmények találhatók például a [De 79], [De 98c] cikkekben).

Az O_2 becslés végleges leírásában a λ_L^+, λ_U^+ és a g függvény helyett az eredeti λ_L, λ_U állandókat és az e függvényt használjuk, vagyis egyidejűleg számítjuk a $g(\mathbf{y})$ és $g(-\mathbf{y})$ függvényértékeket – természetesen ez megfelezi azon vektorok számát, amelyekre a λ_L, λ_U értékeket ki kell számítani. Az O_2 becslést megvalósító algoritmus lényegi lépéseit az alábbiakban adjuk meg.

O_2 algoritmus (általános eset, egy U rendszer esetén)

1. Generáljuk az $U = \{\mathbf{u}^i\}$ rendszert és számítsuk ki az $\overline{\mathbf{u}}^i = T\mathbf{u}^i, i = 1, \ldots, n$ vektorokat.
2. Legyen $Sum = 0$.
3. Határozzuk meg az összes lehetséges $\mathbf{z} = \mathbf{z}^{i,j,\mathbf{s}} = T\mathbf{v}^{i,j,\mathbf{s}} = \frac{1}{\sqrt{2}}\left(s_1\overline{\mathbf{u}}^i + s_2\overline{\mathbf{u}}^j\right)$, $(i, j, \mathbf{s}) \in J^*$ vektorokat és minden $\mathbf{z}$ vektorra végezzük el a 4. lépést.
4. Kezdet: (az $e(\mathbf{z})$ függvény kiszámítása)
   kiszámítjuk a $\lambda_L = \min\{\lambda \mid \lambda\mathbf{z} \in X\}$, $\lambda_U = \max\{\lambda \mid \lambda\mathbf{z} \in X\}$ értékeket,
   ha nincs metszés, akkor legyen $\lambda_L = \lambda_U = 0$,
   ha $\lambda_U \geq \lambda_L \geq 0$, akkor legyen $Sum = Sum + K(\lambda_U) - K(\lambda_L)$,
   ha $\lambda_U \geq 0 \geq \lambda_L$, akkor legyen $Sum = Sum + K(\lambda_U) + K(-\lambda_L)$,
   ha $0 \geq \lambda_U \geq \lambda_L$, akkor legyen $Sum = Sum + K(-\lambda_L) - K(-\lambda_U)$,
   vége (az $e(\mathbf{z})$ függvény kiszámítása).
5. Adjuk át a keresett valószínűség torzítatlan becsléseként a $p = Sum/[2n(n-1)]$ értéket.

Megjegyezzük, hogy egy teljes algoritmusban az U rendszernek N darab független realizációját használjuk, N-et nevezzük a mintaszámnak. Az O_2 becslés kiszámítása azért hatékonyabb a durva módszernél, mert egy U rendszer esetén $O(n^2)$ mátrixszorzás helyett csak $O(n)$ mátrixszorzásra van szükség és még néhány skalár szorzásra. Ez a számítástechnikai hatékonyság növekedés az összes, továbbiakban leírásra kerülő esetben igaz, vagyis eloszlásfüggvény, téglatest, poliéder, ellipszoid, körkúp valószínűségének kiszámítása esetén is.

A (3.7) egyenletben leírt integráltranszformációt és az eredményül kapott kettős integrált többféleképpen is felfoghatjuk.

(i) Tetszőleges $p = \mathbf{P}\{\boldsymbol{\xi} \in X\}$ valószínűség felírható, mint az X halmaz

$$f(\boldsymbol{\xi}) = \begin{cases} 1, & \text{ha } \boldsymbol{\xi} \in X, \\ 0, & \text{egyébként} \end{cases} \tag{3.15}$$

indikátor valószínűségi változójának a várható értéke, vagyis

$$p = \mathbf{E}[f(\boldsymbol{\xi})].$$

Ez a várható érték megfelel a (3.4) jobboldalán álló integrálnak. Felhasználva a $\mathbf{E}(\alpha) = \mathbf{E}[\mathbf{E}(\alpha|\beta)]$ ismételt (feltételes) várható érték összefüggést ez a várható érték átírható a

$$p = \mathbf{E}[f(\boldsymbol{\xi})] = \mathbf{E}[f(\chi_n T\boldsymbol{\eta})] = \mathbf{E}[\mathbf{E}(f(\chi_n T\boldsymbol{\eta}) \mid \boldsymbol{\eta})] \tag{3.16}$$

alakba, ami viszont pontosan megfelel (3.7) kettős integráljának.

(ii) Egy másik lehetséges értelmezés adódik a numerikus integrálás szempontjainak figyelembevételével. A Monte Carlo integrálás elég jól működik, ha a feladat dimenziója nagy, de tudjuk, hogy viszonylag lassú, $O(N^{-1/2})$ a konvergencia sebessége. A hagyományos (determinisztikus) integrálási szabályok kis hibával dolgoznak, de ezeket nem nagyon lehet magasabb dimenzióban használni.

A kettős integrál formájába írt kifejezés a munkánkat két részre osztja: egy egydimenziós, vonal menti integrál meghatározása hagyományos numerikus integrálási technika segítségével és egy, az n-dimenziós térben elhelyezkedő $(n-1)$-dimenziós felületen elvégzett Monte Carlo integrálásra. Ezt a felbontást sugaras-felületi (radial–spherical) integrálásnak [MG 97], vagy iránymenti szimulációnak (directional simulation) is nevezik [DB 89], s a többdimenziós t-eloszlás eloszlásfüggvényének kiszámítására, illetőleg más halmazok valószínűségének meghatározására is használható (elliptikusan szimmetrikus sűrűségfüggvények esetén).

(iii) Végül a Monte Carlo integrálás szempontjából is megvizsgálhatjuk a dekompozíciót. Minden szimuláció esetén a fő kérdés az, hogyan lehet csökkenteni a becslés szórását (anélkül, hogy lényegesen megnövelnénk a szükséges munkát). Ez az eljárás éppen erre példa – a változók számának csökkentésével szóráscsökkenést érünk el. A Monte Carlo integrálás területén szokásos szóhasználattal egy ortonormált becslést egy determinisztikus integrálási formula randomizált változatának is nevezhetünk.

3.3. Egyszerű konvex halmazok

Az előző szakaszból látható, hogy egy, az X halmaz valószínűségének meghatározására a gyakorlatban használható algoritmus előállítható akkor, ha rendelkezésünkre áll a λ_L, λ_U értékek meghatározását elvégző „jó” eljárás, vagyis viszonylag egyszerű lépésekből álló, gyorsan kiszámítható eljárás. Ebben a részben ilyen algoritmusokat adunk meg néhány egyszerű konvex halmazra: lineáris egyenlőtlenségekkel meghatározott poliéder, hiperellipszoid és körkúp esetén. De nemcsak konvex halmazokra alkalmazható az általános elgondolás; például csillag alakú halmazok ([Loh 93]), vagy invertálható függvények segítségével megadott határokkal rendelkező halmazok valószínűsége szintén kiszámítható. Nem-konvex halmazok valószínűsége is meghatározható, ha egy egyenesnek az X halmazba eső részéhez tartozó valószínűségi tartalom megadható (ha például az X halmaz $X = Q_1 \cup Q_2, Q_1 \cap Q_2 = \emptyset$ formában dekomponálható, ahol Q_1, Q_2 a fentebb felsorolt konvex halmazok valamelyike).

A $H = \{\mathbf{x} \mid \mathbf{x} \leq \mathbf{h}\}$ halmaz valószínűségének meghatározásához (az eloszlásfüggvény értékének kiszámításához) a szükséges részleteket [De 80a] tartalmazza, a $Q = \{\mathbf{x} \mid \mathbf{a} \leq \mathbf{x} \leq \mathbf{b}\}$ téglatesttel foglalkoztunk a [De 86b] cikkben. Az alábbiakban megmutatjuk, hogyan lehet λ_L, λ_U értékeket kiszámítani általános poliéder, hiperellipszoid és körkúp esetén.

3.3.1. Konvex poliéderek

Tekintsük véges számú lineáris egyenlőtlenség által adott félterek közös részét, vagyis tegyük fel, hogy a poliédert

$$X = \{\mathbf{x} \mid A\mathbf{x} \leq \mathbf{b}\} \tag{3.17}$$

határozza meg, valamilyen adott $A, \mathbf{b}$ esetén. Feltesszük, hogy X nem üres és nem elfajult (van a halmaznak belső pontja). Ennek megfelelően egy adott $\mathbf{z}$ vektorra a keresett λ_L, λ_U értékek úgy adhatók meg, mint a következő szélsőértékek értékei:

$$\begin{aligned} \lambda_L &= \min\{\lambda \mid A(\lambda\mathbf{z}) \leq \mathbf{b}\}, \\ \lambda_U &= \max\{\lambda \mid A(\lambda\mathbf{z}) \leq \mathbf{b}\}, \end{aligned} \tag{3.18}$$

ha a $\lambda\mathbf{z}$ egyenes elmetszi az X halmazt. Vezessük be a $\mathbf{d} = A\mathbf{z}$ jelölést és egyszerűség kedvéért a $\mathbf{d}$ vektor zéró komponenseit helyettesítsük az $\varepsilon \sim 10^{-10}$ értékkel. Definiáljunk két indexhalmazt, legyen $I_+ = \{i \mid d_i > 0, i = 1, \ldots, n\}, I_- = \{i \mid d_i < 0, i = 1, \ldots, n\}$. Ekkor a megfelelő szélsőértékek:

$$\begin{aligned} \lambda_L &= \max\{b_i/d_i \mid i \in I_-\}, \\ \lambda_U &= \min\{b_i/d_i \mid i \in I_+\}. \end{aligned} \tag{3.19}$$

Ha a $\lambda A\mathbf{z} \leq \mathbf{b}$ feladatnak nincs megengedett megoldása (vagyis $\lambda_L > \lambda_U$), akkor legyen $\lambda_L = \lambda_U = 0$.

A $\mathbf{d} = A\mathbf{z}$ vektorok meghatározásánál – elkerülendő a mátrixszorzásokat minden $\mathbf{z}$ esetén – a következő módosítást kell alkalmazni a végső algoritmusban. A $\mathbf{z}$ vektort mint két U-beli vektor $\mathbf{v}^{i,j,\mathbf{s}} = \frac{1}{\sqrt{2}}(s_1\mathbf{u}^i + s_2\mathbf{u}^j)$ összegének transzformáltját kapjuk. Ezért a $\mathbf{z} = T\mathbf{v} = T\mathbf{v}^{i,j,\mathbf{s}} = \frac{1}{\sqrt{2}}(s_1 T\mathbf{u}^i + s_2 T\mathbf{u}^j)$, tehát az $A\mathbf{z}$ szorzat meghatározásához definiáljuk a $B = AT$ mátrixot és ezzel kapjuk a $\mathbf{d}^{i,j,\mathbf{s}} = A\mathbf{z}^{i,j,\mathbf{s}} = \frac{1}{\sqrt{2}}(s_1 B\mathbf{u}^i + s_2 B\mathbf{u}^j)$ egyenlőséget. Így amikor egy U rendszert generálunk, akkor azonnal kiszámítjuk a $\overline{\mathbf{u}}^i = B\mathbf{u}^i/\sqrt{2}, i = 1, \ldots, n$ vektorokat. Ezek után a $\mathbf{d}^{i,j,\mathbf{s}}$ vektorok meghatározása már csak az $\overline{\mathbf{u}}^i = B\mathbf{u}^i/\sqrt{2}$ vektorok összeadását (kivonását) igényli (az összegnek az $1/\sqrt{2}$ értékkel való normálását is a $\overline{\mathbf{u}}^i$ vektorokra végezzük, hogy a számítási időt csökkentsük). Ennek alapján megint csak n darab $B = AT$-vel való mátrixszorzásra van szükségünk a T és az A mátrixszokkal való $n(n-1)$ mátrixszal való szorzás helyett (ami a durva becslés esetében szükséges lenne).

O_2 algoritmus (poliéder, egy U rendszer esetén).

1. Generáljuk az $U = \{\mathbf{u}^i\}$ rendszert és számítsuk ki az $\overline{\mathbf{u}}^i = B\mathbf{u}^i/\sqrt{2}$, $i = 1, \ldots, n$ vektorokat.
2. Legyen $Sum = 0$.
3. Határozzuk meg az összes lehetséges $\mathbf{d} = \mathbf{d}^{i,j,\mathbf{s}} = s_1\overline{\mathbf{u}}^i + s_2\overline{\mathbf{u}}^j$, $(i, j, \mathbf{s}) \in J^*$ vektort és mindegyik $\mathbf{d}$ esetén végezzük el a következő lépést.
4. Kezdet (a valószínűségi tartalom $e(\mathbf{d})$ függvényének meghatározása) a 3.2 szakaszban leírtakhoz hasonlóan járunk el

vége (az $e(\mathbf{d})$ függvény kiszámításának).
5. Adjuk át a $p = Sum/[2n(n-1)]$ értéket.

Ennek az általános poliéderre vonatkozó algoritmusnak speciális esete az, amikor a $\Phi(\mathbf{h})$ eloszlásfüggvény értékének kiszámítását, vagy egy téglatest valószínűségét akarjuk meghatározni. Természetesen a téglatestek valószínűségének (vagy eloszlásfüggvény értékének) meghatározására kifejlesztett eljárás valamivel gyorsabb, mint az itt leírt általános módszer (lásd [De 86b], [De 98c]), hiszen a belépési és kilépési állandók egyszerűbben számíthatók. Megjegyezzük még, hogy a fenti módon végezhető egy polihedrikus kúp esetén a λ_L, λ_U konstansok meghatározása is, hiszen az is a fenti formájú egyenlőtlenségekkel van megadva.

3.3.2. Hiperellipszoid

Tekintsünk egy B pozitív definit szimmetrikus mátrixot. Ekkor egy $\mathbf{c}$ középpontú, r sugarú nemdegenerált n-dimenziós E ellipszoidot adhatunk meg a következő definícióval:

$$E = \{\mathbf{x} \mid (\mathbf{x}-\mathbf{c})'B(\mathbf{x}-\mathbf{c}) \leq r^2\}.$$

A $\lambda\mathbf{z}$ egyenes akkor metszi el az E hiperellipszoidot, ha a

$$(\lambda\mathbf{z}-\mathbf{c})'B(\lambda\mathbf{z}-\mathbf{c}) = r^2$$

egyenletnek két különböző λ_1, λ_2 gyöke van. Ezeket a lehetséges gyököket az alábbi egyenlet megoldásával lehet megtalálni:

$$\alpha\lambda^2 + \beta\lambda + \gamma = 0, \tag{3.20}$$

ahol az α, β, γ együtthatókat az előző egyenletből lehet meghatározni:

$$\begin{aligned} \alpha &= \mathbf{z}'B\mathbf{z}, \\ \beta &= -2\mathbf{z}'B\mathbf{c}, \\ \gamma &= \mathbf{c}'B\mathbf{c} - r^2. \end{aligned} \tag{3.21}$$

Jelölje a kisebbik gyököt λ_L, a nagyobbik pedig legyen λ_U. Ha a $4(\mathbf{z}'B\mathbf{c})^2 - 4(\mathbf{c}'B\mathbf{c} - r^2)(\mathbf{z}'B\mathbf{z}) < 0$ egyenlőtlenség fennáll, akkor nincs megoldás, vagyis a $\lambda\mathbf{z}$ egyenes nem metszi el a hiperellipszoidot. Egyetlen gyök létezése azt jelenti, hogy az egyenes csak érinti az E ellipszoidot.

A γ értékét csak egyszer kell kiszámítani. A szükséges számítások mennyiségét csökkentendő nem közvetlenül számítjuk ki minden egyes $\mathbf{z}$ vektorra az α, β együtthatókat, hanem egy külön eljárást adunk. Az O_2 becslésnek tetszőleges $\mathbf{z}$ vektorra való alkalmazása esetén $\mathbf{z} = T\mathbf{v}^{i,j,\mathbf{s}} = \frac{1}{\sqrt{2}}(s_1 T\mathbf{u}^i + s_2 T\mathbf{u}^j)$ egyenlőség áll fenn. A B mátrix felbontható egy T_B felső háromszög mátrix segítségével $B = T_B T_B'$ alakban, ekkor a $C = T_B' T$ jelölést bevezetve egy rögzített $i, j, \mathbf{s}$ index hármasra kapjuk, hogy

$$\mathbf{z}'B\mathbf{z} = \frac{1}{2}\sum_{k=i,j}\sum_{l=i,j,l\neq k} s_k s_l \mathbf{u}^k (T'T_B)(T_B'T)\mathbf{u}^l = \frac{1}{2}\sum_{k=i,j}\sum_{l=i,j,l\neq k} s_k s_l (C\mathbf{u}^k)'(C\mathbf{u}^l).$$

Az α együtthatónak ez a formája a $\mathbf{z}'B\mathbf{z}$ kifejezés kiszámításának a következő, számítástechnikailag előnyös módját sugallja: előre számítsuk ki az $\overline{\mathbf{u}}^i = C\mathbf{u}^i, i = 1,\ldots,n,\ r_{ij} = \overline{\mathbf{u}}^{i'}\overline{\mathbf{u}}^j, i,j = 1,\ldots,n, i < j$ értékeket és utána a $\mathbf{z}'R\mathbf{z}$ mennyiséget egyszerűen skalárok összeadásával határozzuk meg:

$$\alpha = \mathbf{z}'B\mathbf{z} = s_1 s_1 r_{ii} + 2s_1 s_2 r_{ij} + s_2 s_2 r_{jj}.$$

A (3.20) másodfokú egyenlet lineáris tagjának β együtthatójára vonatkozóan pedig a következő kifejezést állíthatjuk elő:

$$\mathbf{z}'B\mathbf{c} = \frac{1}{\sqrt{2}}\left(s_1 \mathbf{u}^{i'}T'B\mathbf{c} + s_2 \mathbf{u}^{j'}T'B\mathbf{c}\right).$$

Vezessük be a $\mathbf{d} = T'B\mathbf{c}, \widehat{u}^i = \frac{1}{\sqrt{2}}\mathbf{d}'\mathbf{u}^i, i = 1,\ldots,n$ jelöléseket, ekkor a β együttható értéke szintén skalárok összeadásával kapható meg:

$$\beta = -2\mathbf{z}'B\mathbf{c} = -2(s_1\mathbf{d}'\mathbf{u}^i + s_2\mathbf{d}'\mathbf{u}^j) = -2\left(s_1\widehat{u}^i + s_2\widehat{u}^j\right).$$

A szükséges műveletek száma a következőképpen határozható meg: egy U rendszer (vagyis az O_2 becslésben használt $n(n-1)$ különböző egyenes) esetén a $C\mathbf{u}^i$ értékek kiszámításához csak n darab mátrixszorzásra, az r_{ij} értékek kiszámításához $n(n-1)/2$ skalárszorzathoz, valamint az $\mathbf{d}'\mathbf{u}^i$ kiszámításához n darab skalárszorzatra van szükség – természetesen valamennyi összeadás és némi előkészítés is szükséges (például a $\mathbf{d} = T'B\mathbf{c}, C = T_B'T$ értékek kiszámítása).

O_2 algoritmus (hiperellipszoid, egy U rendszer).
0. Határozzuk meg a $\gamma = \mathbf{c}'B\mathbf{c} - r^2$ értéket.
1. Generáljuk az $U = \{\mathbf{u}^i\}$ rendszert és számítsuk ki az $\overline{\mathbf{u}}^i = C\mathbf{u}^i$, $i = 1,\ldots,n, r_{ij} = \overline{\mathbf{u}}^{i'}\overline{\mathbf{u}}^j, i,j = 1,\ldots,n, i \le j, \widehat{u}^i = \mathbf{d}'\overline{\mathbf{u}}^i, i = 1,\ldots,n$ értékeket.
2. Legyen $Sum = 0$.
3. Határozzuk meg az összes lehetséges $\mathbf{z} = T\mathbf{v}^{i,j,\mathbf{s}}, (i,j,\mathbf{s}) \in J^*$ vektort és mindegyik $\mathbf{z}$ vektorra végezzük el a következő lépést.
4. Kezdet (az $e(\mathbf{z})$ függvény kiszámítása)
   határozzuk meg az α, β együtthatókat, aztán a λ_L, λ_U értékeket,
   folytassuk a valószínűségi tartalom kiszámítását,
   a 3.2 szakaszban leírt módon,
   vége (az $e(\mathbf{z})$ függvény értékének meghatározása)
5. Adjuk át az ellipszoid $p = Sum/[2n(n-1)]$ valószínűségét.

3.3.3. Körkúp

Tekintsünk egy $\mathbf{o}_s$ középpontú és r sugarú, n-dimenziós gömböt, ennek felszinét, valamint ennek egy hipersíkkal való metszetét, vagyis legyen

$$S = \{\mathbf{x} \mid (\mathbf{x}-\mathbf{o}_s)'(\mathbf{x}-\mathbf{o}_s) = r^2,\ \mathbf{n}'(\mathbf{x}-\mathbf{o}_s) = 0\},$$

ahol $\mathbf{n}$ a gömb középpontján átmenő hipersík normálvektora. Jegyezzük meg, hogy S egy $(n-2)$-dimenziós gömbfelület az R^n-ben. Egy általános C körkúpot, melynek csúcsa a $\mathbf{c}$ pontban van, és a felülete tartalmazza az S felületet úgy adhatunk meg, hogy a palástját a következő módon definiáljuk:

$$X = \{\mathbf{y} \mid \mathbf{y} = \mu(\mathbf{x}-\mathbf{c}) + \mathbf{c}, \mathbf{x} \in S, \mu \geq 0\}. \tag{3.22}$$

Egy adott, rögzített $\mathbf{z}$ vektor esetén a $\lambda\mathbf{z}$ egyenesnek az X halmazzal való metszését a következő $n+2$ egyenlet adja meg:

$$\begin{aligned} \mu(\mathbf{x}-\mathbf{c}) + \mathbf{c} &= \lambda\mathbf{z},\ \mu \geq 0, \\ \mathbf{n}'(\mathbf{x}-\mathbf{o}_s) &= 0, \\ (\mathbf{x}-\mathbf{o}_s)'(\mathbf{x}-\mathbf{o}_s) &= r^2 \end{aligned} \tag{3.23}$$

Az első egyenletből kifejezzük az $\mathbf{x} = [\lambda\mathbf{z} - (1-\mu)\mathbf{c}]/\mu$ vektort, a második sorból pedig a $\mu = [\lambda\mathbf{n}'\mathbf{z} - \mathbf{n}'\mathbf{c}]/[\mathbf{n}'\mathbf{o}_s - \mathbf{n}'\mathbf{c}]$ értéket. Ezeket az utolsó sorba helyettesítve λ-ra egy másodfokú egyenletet kapunk:

$$\begin{aligned} & \alpha\lambda^2 + \beta\lambda + \gamma = 0, \text{ ahol} \\ \alpha &= [\mathbf{n}'\mathbf{o}_s - \mathbf{n}'\mathbf{c}]^2\mathbf{z}'\mathbf{z} + [\mathbf{c}'\mathbf{c} + \mathbf{o}_s'\mathbf{o}_s - \mathbf{2c}'\mathbf{o}_s - \mathbf{r^2}](\mathbf{n}'\mathbf{z})^2 + \\ & [2(\mathbf{n}'\mathbf{o}_s - \mathbf{n}'\mathbf{c})](\mathbf{n}'\mathbf{z})(\mathbf{c}'\mathbf{z}) - [2(\mathbf{n}'\mathbf{o}_s - \mathbf{n}'\mathbf{c})](\mathbf{n}'\mathbf{z})(\mathbf{o}_s'\mathbf{z}), \\ \beta &= [2(\mathbf{n}'\mathbf{c})r^2 + 2(\mathbf{n}'\mathbf{o}_s + \mathbf{n}'\mathbf{c})\mathbf{c}'\mathbf{o}_s - \mathbf{2}(\mathbf{n}'\mathbf{o}_s)(\mathbf{c}'\mathbf{c}) + (\mathbf{n}'\mathbf{c})(\mathbf{o}_s'\mathbf{o}_s)]\mathbf{n}'\mathbf{z} + \\ & [-2(\mathbf{n}'\mathbf{o}_s - \mathbf{n}'\mathbf{c})\mathbf{n}'\mathbf{o}_s]\mathbf{c}'\mathbf{z} + [2(\mathbf{n}'\mathbf{o}_s - \mathbf{n}'\mathbf{c})\mathbf{n}'\mathbf{c}]\mathbf{o}_s'\mathbf{z}, \\ \gamma &= (\mathbf{n}'\mathbf{o}_s)^2\mathbf{c}'\mathbf{c} + (\mathbf{n}'\mathbf{c})^2\mathbf{o}_s'\mathbf{o}_s - 2\mathbf{n}'\mathbf{o}_s\,\mathbf{n}'\mathbf{c}\,\mathbf{c}'\mathbf{o}_s - \mathbf{n}'\mathbf{c}\,\mathbf{n}'\mathbf{c}\,r^2. \end{aligned} \tag{3.24}$$

Jelölje a (3.24) egyenlet két megoldását λ_1, λ_2. Ekkor $\mu_1 = [\lambda_\mathbf{1}\mathbf{n}'\mathbf{z} - \mathbf{n}'\mathbf{c}]/[\mathbf{n}'\mathbf{o}_s - \mathbf{n}'\mathbf{c}]$, $\mu_2 = [\lambda_\mathbf{2}\mathbf{n}'\mathbf{z} - \mathbf{n}'\mathbf{c}]/[\mathbf{n}'\mathbf{o}_s - \mathbf{n}'\mathbf{c}]$.

Ha a (3.24) egyenletnek nincs megoldása, akkor az egyenesnek és az X halmaznak nincsen metszete. Ebben az esetben legyen $\lambda_L = \lambda_U = 0$. Ha két különböző λ_1 és λ_2 gyöke van az egyenletnek, akkor a sugár elmetszi a kúpot, de további elemzésre van szükség annak eldöntésére, hogy a $\lambda\mathbf{z}$ egyenesnek melyik része van a kúpon belül. Nem szabad megfeledkezni arról, hogy minden gyök esetén ellenőrizzük a $\mu \geq 0$ nemnegatívitást is; ha egy adott λ_1 gyök esetén például a megfelelő μ_1 negatív, akkor a $\lambda_1\mathbf{z}$ pont a kúp „negatív" részén van, vagyis nincsen valódi metszet – a $\{\lambda\mathbf{z}, \lambda \in [\lambda_2, \infty]\}$ félegyenes van a kúpban. A feladatban szereplő egyenes és a körkúp egymáshoz viszonyított elhelyezkedésétől (vagyis a λ_1, λ_2 gyökök értékeitől és a megfelelő μ_1, μ_2 értékek negatívitásától illetőleg

pozitívitásától) függően négy különböző esetet kell megkülönböztetnünk, ha az origó a kúpon belül van és tizenkét esetet, ha az origó a kúpon kívül van. Hasonlóképpen, ha az utolsó másodfokú egyenlet egy elsőfokúvá degenerálódik, amelynek csak egy λ_0 gyöke van egy $\mu_0 \geq 0$ értékkel, akkor a félegyenes a kúpban van és további vizsgálat szükséges annak eldöntéséhez, hogy $(-\infty, \lambda_0]$ vagy pedig $[\lambda_0, \infty)$ van a C kúpban.

A szükséges számítástechnikai munka mennyiségét csökkentjük a következő megfontolások alapján. Az α, β, γ együtthatókban szereplő skalárszorzatokat három csoportba oszthatjuk. Az első csoport tartalmazza azokat a szorzatokat, amelyek a konstans $\mathbf{n}, \mathbf{c}, \mathbf{o}$ vektorok szorzataként jelennek meg, ezeket csak egyszer kell meghatározni. A második csoportba tartoznak a $\mathbf{c}'\mathbf{z}, \mathbf{n}'\mathbf{z}, \mathbf{o}'\mathbf{z}$ skalárszorzatok, a harmadik csoport pedig a $\mathbf{z}'\mathbf{z}$ szorzat – ezek a $\mathbf{z}$ vektortól függően változnak. Megmutatjuk, hogy az O_2 becslés használata esetén hogyan lehet a számítási munkát csökkenteni a $\mathbf{z}'\mathbf{z}$ szorzat esetén, valamint a második csoportból egy szorzat, a $\mathbf{n}'\mathbf{z}$ kiszámítása esetén (a második csoport többi szorzata hasonlóképpen elintézhető).

Vezessünk be új jelöléseket, legyen $\overline{\mathbf{u}}^i = T\mathbf{u}^i$ és legyen $\mathbf{z}^{i,j,\mathbf{s}} = \frac{1}{\sqrt{2}}(s_1 T\mathbf{u}^i + s_2 T\mathbf{u}^j)$ ekkor $\widehat{u}^i = \mathbf{n}'\overline{\mathbf{u}}^i, i = 1, \ldots, n, r_{ij} = \overline{\mathbf{u}}^{i'}\overline{\mathbf{u}}^j, i, j = 1, \ldots, n$. Ezekkel a jelölésekkel a két keresett skalárszorzat felírható mint skalárok összege: $\mathbf{n}'\mathbf{z} = \frac{1}{\sqrt{2}}(s_1\widehat{u}^i + s_2\widehat{u}^j)$ (ugyanezt kell végrehajtani a $\mathbf{c}'\mathbf{z}, \mathbf{o}'\mathbf{z}$ szorzatokra is) és $\mathbf{z}'\mathbf{z} = s_i s_i r_{ii} + 2s_i s_j r_{ij} + s_j s_j r_{jj}$.

Az előző részekben leírtakhoz hasonlóan a műveletek megfelelő csoportosításával és átrendezésével kaphatjuk a következő eljárást:

O_2 algoritmus (körkúp, egy U rendszer).
0. Számítsuk ki a $\mathbf{n}'\mathbf{c}, \mathbf{n}'\mathbf{o}_s, \mathbf{c}'\mathbf{o}_s, \mathbf{o}_s{}'\mathbf{o}_s, \mathbf{c}'\mathbf{c}$ szorzatokat.
1. Generáljuk az $U = \{\mathbf{u}^i\}$ rendszert és utána számítsuk ki az $\overline{\mathbf{u}}^i = T\mathbf{u}^i$, $\widehat{u}^i = \mathbf{n}'\overline{\mathbf{u}}^i, i = 1, \ldots, n$ mennyiségeket, majd a többi skalárszorzatot is a második és harmadik csoportból, végül pedig az r_{ij} értékeket.
2. Legyen $Sum = 0$.
3. Határozzuk meg az összes lehetséges $\mathbf{z} = \mathbf{z}^{i,j,\mathbf{s}}$, $(i, j, \mathbf{s}) \in J^*$ vektort, és minden egyes $\mathbf{z}$ vektorra végezzük el a következő lépést.
4. Kezdet (a vektor valószínűségi tartalmának meghatározása) egy adott $i, j, \mathbf{s}$ indexhármas esetén határozzuk meg az α, β, γ skalárokat a következő lépések végrehajtásával: először meghatározzuk az $r_{ij}, \widehat{u}^j$ értékeket, majd ezek segítségével számítsuk ki a $\mathbf{z}'\mathbf{z}, \mathbf{n}'\mathbf{z}, \ldots$ értékeket, majd a $\lambda_L = \min\{\lambda \mid \lambda\mathbf{z} \in X\}$ és a $\lambda_U = \max\{\lambda \mid \lambda\mathbf{z} \in X\}$ számokat a λ_1, λ_2 gyökök segítségével, végül a μ_1, μ_2 előjelének figyelembevételével az egyenes valószínűségét adjuk a Sum változóhoz, vége (a valószínűségi tartalom meghatározása).
5. Adjuk át a valószínűség értékeként a $p = Sum/[2n(n-1)]$ értéket.

Az iránymenti integrálás eljárásának minden egyes $\mathbf{z}$ vektorra való közvetlen alkalmazásával $n(n-1)$ darab $T(\lambda\mathbf{z})$ mátrixszorzásra lenne szükségünk. A fentebbi algorit-

musban javasolt számítási elrendezésben csak az $\overline{\mathbf{u}}^i = T\mathbf{u}^i$ mennyiségekhez szükséges n mátrixszorzást kell elvégezni, továbbá $\widehat{u}^i = \mathbf{n}'\overline{\mathbf{u}}^i$ mennyiségekhez n skalárszorzás, valamint az $r_{ij} = \overline{\mathbf{u}}^{i'}\overline{\mathbf{u}}^j$ mennyiségekhez szükséges $n(n-1)/2$ skalárszorzás kell, vagyis még egy mátrixszorzásnyi művelet.

Ha egy polihedrikus kúp nagyon sok egyenlőtlenséggel van megadva, akkor érdemes lehet két körkúpot konstruálni, amelyek közül az egyik tartalmazza a polihedrikus kúpot, a másik pedig benne van. A két körkúp segítségével korlátokat adhatunk a polihedrikus kúp valószínűségére.

3.4. Általános n-dimenziós halmazok

3.4.1. A valószínűségek korlátozása

A fentebb megvizsgált tartományokon kívül – más alkalmazásokban – másmilyen halmazok valószínűségének kiszámítására lehet szükség. A megbízhatósági feladatokban a csőd tartományának (lásd például [Bre 94], [DB 89], és a [De 98a] cikket tartalmazó kötet) valószínűségét keressük. Ezt a tartományt általában egyenlőtlenségek adják meg:

$$X = \{\mathbf{x} \mid g_i(\mathbf{x}) \leq 0, i = 1, \ldots, M\}, \tag{3.25}$$

valamilyen g_i függvényekkel, amelyeket nehéz kiszámítani. Ha a λ_L, λ_U belépési és kilépési állandók könnyen meghatározhatók, akkor az ortonormalizált eljárás nehézség nélkül alkalmazható. Nyilván az i-edik feltétel határát azon λ_i esetén metszi el a $\lambda\mathbf{z}, \lambda \geq 0$ félegyenes, amelyre $g_i(\lambda_i\mathbf{z}) = 0$ fennáll. Tehát a belépési és kilépési állandók meghatározhatósága a

$$\lambda_L = \min\{\lambda \mid g_i(\lambda\mathbf{z}) = 0, \lambda \geq 0, i = 1, \ldots, M\},$$
$$\lambda_U = \max\{\lambda \mid g_i(\lambda\mathbf{z}) = 0, \lambda \geq 0, i = 1, \ldots, M\}$$

feladatok megoldhatóságától függ. Tekintsük az $\mathbf{x}'Q\mathbf{x} + \mathbf{b}'\mathbf{x} + c \leq 0$ kvadratikus feltételt, ahol Q egy pozitív definit mátrix. Az $\mathbf{x}'Q\mathbf{x} + \mathbf{b}'\mathbf{x} + c = 0$ felületnek és a $\lambda\mathbf{z}, \lambda \geq 0$ sugárnak a metszéspontjait nyilván a $\lambda^2\mathbf{z}'Q\mathbf{z} + \lambda\mathbf{b}'\mathbf{z} + c = 0$ egyenlőség adja, amelyből a λ_1, λ_2 gyökök kifejezhetők és a továbbiak vonatkozásában ugyanúgy járhatunk el, mint például az ellipszoidnál. Ezek szerint a kvadratikus feltételek esetén az ortonormált becslések alkalmazhatók, ezért egy általános, kétszer differenciálható függvénnyel felírt $g(\mathbf{x}) \leq 0$ feltétel esetén a következő eljárás segítségével lehet a $p = P\{\boldsymbol{\xi} \in \{\mathbf{x} \mid g(\mathbf{x}) \leq 0\}\}$ valószínűségre közelítéseket kapni. Sorbafejtjük a $g(\mathbf{x})$ függvényt a kvadratikus tagig, majd az így kapott kvadratikus közelítés esetére meghatározzuk a valószínűséget, amely a keresett p valószínűség egy közelítése lesz.

Más esetekben segítséget nyújthat az, ha két olyan X_1, X_2 konvex halmazt tudunk konstruálni (amelyek akármelyik fent említett típusból származhatnak), amelyek közül az egyik tartalmazza X-et, a másik pedig teljesen benne van, vagyis $X_1 \subset X \subset X_2$. Ilyen esetekben az eredeti halmaz valószínűségét a két konvex halmaz valószínűségével

korlátozhatjuk: $Pr(X_1) \leq Pr(X) \leq Pr(X_2)$. Végül, mint említettük, ha az X halmaz előállítható fentebbi típusú, diszjunkt konvex halmazok egyesítéseként, akkor is meg tudjuk határozni a halmaz valószínűségét.

3.4.2. A hatékonyság növelése

Megbízhatósággal kapcsolatos feladatokban a $Pr(X)$ általában kicsi, mivel ez a rendszer tönkremenésének valószínűsége, 0.001-0.05 körül van, vagy még ennél is kisebb. Ennek következményeként a fentebb leírt Monte Carlo módszerek nem nagyon hatékonyak, hiszen a generált vektorok túlnyomó többsége nem metszi az X halmazt. Ha a megoldások néhány (vagy összes) komponensére még nem-negatívitási feltétel is elő van írva, akkor lehetőség van a hatékonyság növelésére – az X halmazt metsző vektorok arányát megnöveljük, aztán a végső eredmény megadásánál súlyokkal kompenzálunk a feltételes valószínűségnek megfelelően.

Tegyük fel, hogy egy rögzített $I = \{i_1, \ldots, i_k\} \subset \{1, 2, \ldots, n\}$ indexhalmaz esetén az n-dimenziós X halmaz a következőképpen van adva:

$$X_I = \{\mathbf{x} \mid g_i(\mathbf{x}) \leq 0, i = 1, \ldots, M, x_j \geq 0, j \in I\}. \tag{3.26}$$

Jelölje az R^n egy alterét $R_I^n = \{\mathbf{x} \mid x_j \geq 0, j \in I\}$ és legyen ennek a valószínűsége a φ sűrűségfüggvény esetén $p_I = Pr\{R_I^n\}$. Vegyük észre, hogy ezt a p_I értéket könnyen meghatározhatjuk (egy adott $\mathbf{z} \in R^n$ vektor esetén a $\mathbf{z} \in R_I^n$ reláció eldöntéséhez csak a generált $\mathbf{z}$ vektor komponenseinek előjelét kell megvizsgálni).

Legyen $Tr : R^n \to R_I^n$ egy transzformáció, amelyet a $Tr(\mathbf{z}) = \overline{\mathbf{z}}$, $\mathbf{z} \in R^n, \overline{\mathbf{z}} \in R_I^n$ összefüggés definiál, ahol

$$\begin{aligned} \overline{z}_i &= |z_i|, \text{ ha } i \in I, \\ \overline{z}_i &= z_i, \quad \text{egyébként.} \end{aligned}$$

Ekkor a Tr segítségével az összes R^n-ben generált vektort R_I^n-be transzformáljuk; a $Pr\{\mathbf{x} \in X_I \mid \mathbf{x} \in R_I^n\}$ feltételes valószínűséget pedig az ortonormált becslések segítségével számítjuk ki (amelyhez az R_I^n-be transzformált vektorokat használjuk). Végül az eredeti halmaz valószínűségét $Pr(X) = p_I Pr\{\mathbf{x} \in X_I \mid \mathbf{x} \in R_I^n\}$ adja meg.

O_2 algoritmus (nemnegatív komponensek esete, egy U rendszer).

1. Generáljuk U-t, és számítsuk ki a $\overline{\mathbf{u}}^i = T\mathbf{u}^i, i = 1, \ldots, n$ vektorokat.
2. Határozzuk meg az $\mathbf{z}^{i,j,\mathbf{s}} = \frac{1}{\sqrt{2}}\left(s_1\overline{\mathbf{u}}^i + s_2\overline{\mathbf{u}}^j\right), (i, j, \mathbf{s}) \in J^*$ vektorokat.
3. Legyen $p_I = 0, Sum = 0$. Minden lehetséges $\mathbf{z}$ vektorra végezzük el a következő lépést:
4. Kezdet (a feltételes valószínűség kiszámítása)
   ha $\mathbf{z} \in R_I^n$, akkor növeljük $p_I = p_I + 1$ és legyen $\overline{\mathbf{z}} = \mathbf{z}$,
   ha $\mathbf{z} \notin R_I^n$, akkor transzformáljuk $\mathbf{z}$-t az R_I^n-be: legyen $\overline{\mathbf{z}} = Tr(\mathbf{z})$,

adjuk hozzá az egyenes valószínűségét $Sum = Sum + g(\overline{\mathbf{z}})$
vége (a feltételes valószínűség kiszámítása)
5. Végül adjuk át a $Pr(X) = Sum/[2n(n-1)] \times p_I/[2n(n-1)]$ értéket.

3.4.3. Egyéb esetek

Egy ortáns valószínűségét egyszerűen kiszámíthatjuk azáltal, hogy egy ortonormált U rendszer generálása után az $\overline{\mathbf{u}}^i$ vektorok előjeleit ellenőrizzük (vagy pedig O_2 becslés esetén $\overline{\mathbf{u}}^i + \overline{\mathbf{u}}^j$ vektorok előjeleit vizsgáljuk).

Ha egy nem zérus várható értékű normális eloszlás esetén van szükségünk az ortáns valószínűségének meghatározására, akkor egy megfelelő eltolás segítségével a normális eloszlás nulla várható értékűvé tehető és az ortáns valószínűség kiszámításának feladata pedig egy eloszlásfüggvény értékének meghatározásává válik, tehát a feladat a fentebbiekben tárgyalt módon megoldható.

Végezetül egy további halmaztípust említünk: tekintsük az úgynevezett csillag-alakú halmazokat (lineáris vagy más feltételek által határolt, az origó körüli halmazok, amelyek valószínűségének meghatározására Lohr adott eljárást [Loh 93]). Ha a csillag-alakú halmaz szimmetrikus az origóra, akkor további hatékonyság-növelést lehet elérni, ha nem szimmetrikus, akkor konvex halmazokra való felbontást használhatunk.

3.5. Számítógépes eredmények

3.5.1. A NORSET számítógépes szubrutinrendszer

A kifejlesztett algoritmusokat FORTRAN nyelvű szubrutinokkal valósítottuk meg számítógépen. A teljes programcsomag 25 szubrutint tartalmaz, a forrásnyelvi program összesen 4100 sorból áll. A teljes rendszert NORSET-nek neveztük el, mert szubrutinjai segítségével eloszlásfüggvény értékeket, téglalapok, poliéderek, ellipszoidok és körkúpok valószínűségét lehet kiszámítani.

A valószínűséget kiszámító szubrutinok lényegében kétféle módon használhatók: vagy megadjuk a mintaszámot, vagy pedig megkövetelünk egy adott pontosságot – ekkor a program határozza meg a szükséges mintaszámot. A hiba nagyságának mérésére egyszerűség kedvéért a becslés szórását használtuk, bár a három szórásnyi hibát szokták általában használni.

A szubrutinrendszer öt tizedes pontosságot biztosít megfelelő paraméterértékek esetén, a hatodik tizedes pontossága kérdéses (még akkor is, ha elég nagy mintaszámot adunk meg). A közölt számítógépes eredmények azt mutatják, hogy három tizedesre pontos eredményt $n = 20$ dimenzióig fél másodpercnél rövidebb idő alatt lehet kapni, míg 100 dimenzióig 5 sec időnél kevesebbre van szükség.

3.5.2. A számítógépes kisérletek részletei

A számítógépes kisérleteket egy IBM RS 6000 gépen végeztük el, az University of Zürich, Institute of Operations Research intézménynél folytatott kutatásaink alatt. A programokat f77 fordítóval használtuk, ahol a standard beállításokat nem változtattuk meg.

A szubrutinok ellenőrzését és viselkedésük jellemzését nagyszámú numerikus példán való lefuttatásával végeztük. A többdimenziós, nem-elfajult normális eloszlás paramétereit vagy rögzítettük, vagy véletlenszerűen generáltuk. Az egyes tartományokat illetően is kétféle feladatot használtunk: mintegy 30 rögzített paraméterű példán (amelyek valószínűségei ismertek [De 90a]) és nagyszámú véletlenszerűen generált feladaton végeztük el a próbákat. A determinisztikus feladatok esetében ismert volt a keresett valószínűség is. A teszteléshez használt feladatok előállításának módját, valamint egyes feladatok numerikus adatait a [De 98c], [De 03a], [De 00] cikkekben írtuk le. A próbafeladatok meghatározásánál a legnagyobb nehézségeket az okozta, hogyan lehet olyan véletlen elhelyezkedésű konvex testet generálni, amelynek a valószínűsége nem nulla, sőt, egy adott valószínűséget megközelít – vagyis azzal a közismert ténnyel kellett szembesülni, hogy az n-dimenziós tér viszonylag „üres".

Mint említettük, az egyes módszerek hibáját a szórással, vagyis a becslés empirikus szórásnégyzetének négyzetgyökével jellemeztük. Természetesen kis mintaszámok esetén ez a hibabecslés megbízhatatlanná vált. A hiba így a N mintaszám négyzetgyökével arányos.

Az egyes algoritmusok hatékonyságát a Monte Carlo módszerek esetén szokásos módon végeztük. Ha az $Algoritmus_1$ és az $Algoritmus_2$ esetén $t_i, i = 1, 2$ időre van szükség a σ_i szórású becslés kiszámításához, akkor a $Algoritmus_2$-nek $Algoritmus_1$-hez viszonyított hatékonyságát a

$$hat. = (t_1/t_2) * \left[\frac{\sigma_1}{\sigma_2}\right]^2$$

hányados adja meg, hiszen ez az aránya az azonos pontosságú (szórású) eredmény eléréséhez szükséges időknek. A számítógépes futásokban mindig a durva módszerhez viszonyítottuk az ortonormált becsléseket.

A hatékonyság értéke nem függött lényegesen a tartománytól, amelynek a valószínűségét akarjuk kiszámítani (a kivétel a poliéder, amely a feltételek számától függően változó hatékonyságot mutatott). Tipikus hatékonyságokat mutatunk be a 3.1 táblázatban.

Megadjuk a táblázatokban található jelölések magyarázatát: n a dimenziószám, $p \sim$ a valószínűség körülbelüli értéke, p_{ex} a pontos valószínűség (ha ismert), p a számított (becsült) érték, σ az eredmény szórása (a hiba), N a mintaszám, t a futási idő másodpercben, $hat.$ pedig az adott módszernek a durva módszerhez viszonyított hatékonysága. Mivel a futási idők lényegében a dimenzió számától és a valószínűségtől függenek, ezért általában ezek függvényeként adjuk meg a futási időket.

	$p \sim 0.40$					$p \sim 0.90$				
	eloszl.	tégl.	polié.	ell.	kúp	eloszl.	tégl.	polié.	ell.	kúp
n= 5	15	15	24	34	10	65	84	125	475	370
n=10	18	15	28	76	17	59	28	29	154	115
n=15	11	9	17	50	18	34	18	14	73	74
n=20	9	29	22	31	43	13	19	12	60	28

3.1. táblázat. Az O_2 becslések hatékonyságai.

$p \sim$	n= 5	$p \sim$	n=10	$p \sim$	n=15	$p \sim$	n=20
0.02	0.2 sec	0.01	0.0 sec	0.01	0.5 sec	0.01	0.7 sec
0.05	0.8 sec	0.07	0.6 sec	0.05	5.0 sec	0.09	19.2 sec
0.12	0.4 sec	0.29	1.8 sec	0.09	10.6 sec	0.38	53.1 sec
0.40	3.0 sec	0.40	2.5 sec	0.40	17.0 sec	0.54	27.6 sec
0.64	2.9 sec	0.55	0.7 sec	0.56	24.1 sec	0.64	37.0 sec
0.80	0.3 sec	0.87	0.3 sec	0.73	9.9 sec	0.78	20.3 sec
0.90	0.3 sec	0.91	0.5 sec	0.88	3.2 sec	0.90	10.5 sec
0.90	0.2 sec	0.93	0.3 sec	0.96	2.0 sec	0.97	3.2 sec
0.99	0.01 sec	0.99	0.0 sec	0.99	0.6 sec	0.99	1.5 sec

3.2. táblázat. Számítási idők másodpercben eloszlásfüggvény esetén, előírt pontosság 10^{-4}.

3.5.3. Futási eredmények táblázatokban

A számítógépes kisérletezések szerint a módszerek működése (futási ideje) lényegesen függött a kiszámítandó valószínűség értékétől – ennek magyarázatát a [De 00] cikkben találhatjuk. Ezért a táblázatokban még a véletlenszerűen generált feladatoknál is feltüntettük a valószínűség értékét a táblázatok fejlécében. Tapasztalatainkat összefoglalva mondhatjuk, hogy a legjobb (leggyorsabb) eredményeket $p < 0.05$ és $0.9 < p < 1.0$ esetekben kaptuk, míg a legrosszabb eredményeket $p \sim 0.4$ esetén kaptuk.

Az alábbi 3.2 – 3.6 táblázatokban közöljük az egyes feladattípusokra vonatkozó futási időket. A futási idők másodpercben vannak megadva arra az esetre, amikor a megkívánt pontosság 10^{-4} (ilyenkor egy kis elemszámú mintával a megfelelő szubrutin meghatározza a szükséges N mintaszámot). A táblázatok az eloszlásfüggvény értékének meghatározása, téglalap, poliéder, ellipszoid és körkúp valószínűségének kiszámítása esetére tartalmazzák a megfelelő adatokat. Megjegyezzük, hogy a poliéderek esetében a számítási idők (nyilvánvalóan) függenek a poliédert meghatározó hipersíkok számától, ezért a poliédereknél kétféle feladatot is előállítottunk; az egyik fajta esetén $n+1$ hipersík adta meg a poliédert, a másiknál pedig $5n - 1$ hipersíkot használtunk, a $felt.$ = mutatja a feltételek számát.

	$p \sim 0.03$	$p \sim 0.50$	$p \sim 0.90$	$p \sim 0.98$
n= 5	0.2 sec	2.7 sec	0.3 sec	0.1 sec
n=10	1.0 sec	8.5 sec	1.2 sec	0.5 sec
n=15	2.8 sec	15.1 sec	3.5 sec	2.7 sec
n=20	3.3 sec	14.3 sec	7.3 sec	4.0 sec

3.3. táblázat. Számítási idők másodpercben téglatestek esetén, előírt pontosság 10^{-4}.

	$p \sim 0.05$		$p \sim 0.40$		$p \sim 0.90$		$p \sim 0.98$	
$felt.=$	$n+1$	$5n-1$	$n+1$	$5n-1$	$n+1$	$5n-1$	$n+1$	$5n-1$
n= 5	1.6	1.3	2.8	1.9	0.2	0.5	0.3	0.0
n=10	2.4	3.1	8.6	13.5	2.1	3.1	0.2	0.2
n=15	4.5	2.5	16.0	38.1	8.5	13.2	0.8	1.3
n=20	1.7	–	22.6	106.2	10.4	35.6	1.2	2.6

3.4. táblázat. Számítási idők másodpercben poliéderek esetén, előírt pontosság 10^{-4}.

	$p \sim 0.05$	$p \sim 0.40$	$p \sim 0.90$	$p \sim 0.98$
n= 5	0.5	2.8	0.1	0.0
n=10	0.1	3.2	0.8	0.1
n=15	4.9	10.8	3.2	0.6
n=20	4.0	27.6	4.9	0.5

3.5. táblázat. Számítási idők másodpercben ellipszoidok esetén, előírt pontosság 10^{-4}.

	$p \sim 0.05$	$p \sim 0.40$	$p \sim 0.90$	$p \sim 0.98$
n= 5	0.5	5.0	0.6	0.0
n=10	1.8	8.4	0.5	0.2
n=15	0.1	15.6	1.5	0.4
n=20	2.0	30.1	4.9	0.5

3.6. táblázat. Számítási idők másodpercben körkúpok esetén, előírt pontosság 10^{-4}.

típus	dim.	p	hiba	N	idő
polié. 11	n=10	0.000070	0.000038	14	0.018 sec
polié. 21	n=20	0.000239	0.000036	12	0.125 sec
polié. 49	n=10	0.000079	0.000014	4	0.058 sec
polié. 99	n=20	0.000846	0.000027	7	0.511 sec
ellips.	n=10	0.002286	0.000101	316	0.063 sec
kúp	n=15	0.001098	0.000102	241	0.120 sec

3.7. táblázat. Kis valószínűségek kiszámítása.

Összefoglalva eredményeinket elmondhatjuk, hogy a legrosszabb ($p \sim 0.4$ körül mért) idők egy ötödére – egy tizedére volt csak szükség akkor, ha a kiszámítandó valószínűségek értéke 0.03 vagy 0.90 körül volt, és a legrosszabb esetben mért idők egy tizedére – egy századára volt csak szükség 0.01 illetőleg 0.98 körüli valószínűségek esetén. A hatékonyságok hasonló arányban megnőttek ezekre a nagyon kicsi, illetőleg nagyon nagy valószínűségű példákra, de az ezres hatékonyságot csak ritkán érték el az algoritmusok.

Külön megvizsgáltuk a megbízhatóság elméletében előforduló, különböző szerkezetek megbízhatóságára vonatkozó számításokban a szerkezet összetörésének és hasonlóan fontos kis valószínűségek ($p < 0.001$) kiszámításának esetét, ezekre vonatkozó eredmények találhatók a 3.7 táblázatban. A poliédereknél megadtuk a definiáló hipersíkok számát. A hiba a becslés empirikus szórását mutatja, vagyis az eredmények szerint a hiba a kiszámítandó valószínűség értékének egy tizede körül van, vagy kisebb, még ezekre a kis p értékekre is. Az N mintaszám természetesen a generált U ortonormált rendszerek száma.

Végezetül még egy kérdéskört vizsgáltunk meg; mi történik a dimenziószám növelése esetén. Erre vonatkozó adatokat adunk meg az utolsó táblázatban az eloszlásfüggvények kiszámítása esetén (a konvex testekre hasonló eredményeket kaptunk). Amint az várható volt, a dimenziószám növelésével a hatékonyság csökken. Mindazonáltal számítási eredményeink alapján még $n = 100$ dimenzióban is ki lehet számítani elfogadható idő alatt három tizedes pontossággal a normális valószínűségeket. □

		$p \sim 0.01$	$p \sim 0.30$	$p \sim 0.90$	$p \sim 0.98$
n= 20	p	0.00043	0.32436	0.85730	0.97772
t=0.2 sec	p_{ex} =	0.00033	0.32526	0.85806	0.97807
	σ=	0.00005	0.00150	0.00074	0.00036
	$hat.$ =	397	5	13	9
n= 40	p	0.01170	0.27598	0.89546	0.99685
t= 1 sec	p_{ex} =	0.01191	0.27502	0.89515	0.99672
	σ=	0.00020	0.00100	0.00066	0.00013
	$hat.$ =	8	5	5	21
n= 60	p	0.00271	0.32026	0.91188	0.99763
t= 4 sec	p_{ex} =	0.00294	0.32211	0.92974	0.99560
	σ=	0.00007	0.00061	0.00053	0.00048
	$hat.$=	46	9	3	1
n= 80	p	0.01500	0.41527	0.88817	0.97652
t= 7 sec	p_{ex} =	0.01493	0.41529	0.88861	0.97608
	σ=	0.00018	0.00068	0.00058	0.00031
	$hat.$ =	5	8	4	3
n=100	p	0.01829	0.20381	0.86847	0.97865
t=13 sec	p_{ex} =	0.01828	0.20282	0.86819	0.97835
	σ=	0.00018	0.00058	0.00049	0.00028
	$hat.$ =	15	6	7	1

3.8. táblázat. Eloszlásfüggvény értékek kiszámítása $20 \leq n \leq 100$ dimenzióban, rögzített $N = 100$ mintaszámra.

4. fejezet

A normális eloszlás egydimenziós közelítései

A fejezet első szakaszában néhány, numerikusan könnyen kiszámítható függvényt adunk meg, amelyek az egydimenziós normális eloszlás eloszlásfüggvényét közelítik. Ezeket a közelítő függvényeket a legkisebb négyzetek módszerével határozzuk meg, és csak egy kvadratikus kifejezés és egy exponenciális függvény kiszámítására van szükség [De 97a].

A 4.2 szakaszban a többdimenziós normális eloszlás eloszlásfüggvényének egy egyenes mentén felvett értékeire (egydimenziós függvényére) adunk meg közelítő függvényeket a [De 98b] cikk alapján. Itt is olyan közelítéseket konstruálunk, amelyek vagy kvadratikus függvények, vagy ezeknek egyszerű transzformációi. A fejezet első részében leírtaktól abban különböznek az itt leírt algoritmusok, hogy most a függvényértékeket nem tudjuk pontosan kiszámítani, egy additív zaj is megjelenik. Ezért ezeket a közelítő függvényeket regressziós becsléseknek nevezzük. A második rész végén megmutatjuk, hogy bizonyos, a logaritmikus transzformáció miatt fellépő torzításokat hogyan lehet ellensúlyozni.

A fejezet harmadik szakaszában megmutatjuk, hogy a 4.2-ben konstruált regressziós becsléseket hogyan lehet közelítő gyök (kvantilis) megkeresésére, illetőleg a gradiens kiszámítására használni [De 98b]. Numerikus tapasztalatokról is beszámolunk – további numerikus eredmények a [De 98a] cikkben találhatók.

4.1. Az egydimenziós normális eloszlásfüggvény közelítései

Gyakran van szükség a normális eloszlás eloszlásfüggvényének értékeire és időnként hasznos, ha egy könnyen kiszámítható approximáló függvény segítségével ki tudjuk számítani legalább közelítőleg ezeket az értékeket. Jónéhány közelítés ismert, némelyek csak nagy hibával adják meg a keresett értéket, míg a bonyolultabbak 6-8 tizedesre pontos eredményt adnak. Ezeknek a közelítéseknek egy jó áttekintése található például a [JK 72] könyvben.

Egy jellemző példa az egyszerű közelítésekre a Hoyt által ajánlott: tekintsünk három,

a $[-1,1]$ intervallumban egyenletes eloszlású valószínűségi változót, vegyük ezek konvolúcióját, és közelítésként ennek az összegnek az eloszlásfüggvényét használjuk. A Hastings, Zelen és Severo által ajánlott közelítések a sűrűségfüggvény változójának egy harmadhatodfokú polinomját használják. Raab és Green közelítő sűrűségfüggvényként a $\frac{1}{2\pi}(1+\cos x)$ függvénynek a használatát javasolják. Egy lényegesen több munkával járó, még jobb közelítés sorbafejtést használ. Például egy hatodfokú polinommal, egy exponenciális és néhány elemi függvény kiszámításával 6-7 tizedesre pontos eredményt lehet kapni az általában könnyen elérhető FORTRAN nyelvű szubrutinkönyvtárakban lévő szubrutinnal. Az alábbiakban megadott közelítések vagy egyszerűbbek, vagy valamivel pontosabbak, mint a fentebb említett közelítések.

Ajánlott közelítéseinket a legkisebb négyzetek módszerével határozzuk meg. A 4.1 szakaszban azt vizsgáljuk meg, hogy milyen közelítést lehet kapni az egy-dimenziós eloszlás esetében. Bár ezek önmagukban is érdekesek, de igazi szerepük a következő rész eredményeinek előkészítése. (A 4.2 szakaszban hasonló módon kiszámított, ugyanilyen alakú függvényeket használunk a többdimenziós normális eloszlás eloszlásfüggvényének az egy egyenes mentén való közelítésére is, ezek is viszonylag jó eredményeket adnak.) A közelítések meghatározásában egyszerre két, egymásnak némileg ellentmondó célt szeretnénk megvalósítani: legyen a formula egyszerű és legyen a közelítés lehetőleg jó. A két cél közötti egyensúly elérésének egy lehetséges kivitelezése a leírt módszer, – bár a kapott approximációk nem a legpontosabbak, de a felhasznált műveletek fajtáját és számát tekintve viszonylag pontosak. A közelítések hibája (az egydimenziós normális eloszlásfüggvénytől való eltérésük) 0.001 és 0.00001 között van. A konstrukcióinknak az az előnye is megvan, hogy ha egy adott feladat megoldása során csak egy adott intervallumban van szükségünk viszonylag pontos közelítésre, akkor az adott intervallumban olyan közelítést lehet előállítani, ami az általános, egész számegyenesen használható közelítésnél jobb approximáló függvényt határoz meg.

A következő 4.1.1 részben a legkisebb négyzetek módszerével meghatározható közelítések általános leírását adjuk, míg a közelítő függvények, az állandók értékeire is kiterjedő teljes leírását a 4.1.2 részben adjuk meg.

4.1.1. Közelítések a legkisebb négyzetek módszerével

Legyen az egydimenziós standard normális eloszlás $\Phi(x)$ eloszlásfüggvénye és $\phi(z)$ sűrűségfüggvénye a szokásos módon adva:

$$\begin{aligned} \Phi(x) &= \int_{-\infty}^{x} \phi(z)\,dz, \\ \phi(z) &= (2\pi)^{-1/2}\, e^{-z^2/2}, z \in (-\infty,\infty). \end{aligned} \tag{4.1}$$

Tegyük fel, hogy adott $x_i, i = 1, \cdots, n$ esetén a $p_i = \Phi(x_i)$ értékeket ki lehet számítani pontosan. A legkisebb négyzetek módszerét használjuk egy $\Phi(x)$ függvényt közelítő $g(x)$ függvény meghatározására. Ez ekvivalens a

$$\min \sum_{i=1}^{n} [p_i - g(x_i)]^2 \tag{4.2}$$

feladat megoldásával, ahol a minimumot a $g(x)$ függvény összes lehetséges (ismeretlen értékű) paraméterén kell venni. A (4.2) feladat megoldását a minimalizálási probléma elsőrendű szükséges optimalizálási feltételeinek felírásával lehet meghatározni, vagyis deriváljuk a minimalizálandó feladatban szereplő összeget az ismeretlen paraméterek szerint, ezeket nullával egyenlővé téve egy egyenletrendszert kapunk, amelyet a paraméterekre megoldunk. A továbbiakban ezt az általános eljárást használjuk a közelítések paramétereinek meghatározására.

A lehetséges approximálásokra három különböző felépítésű függvényt vizsgálunk meg. Jelölje $t_1(x)$ a $\Phi(x)$ eloszlásfüggvény közelítésére kialakított első függvényt, melynek formája egy $g_1(x) = a_1x^2 + b_1x + c_1$ kvadratikus függvény, ahol az a_1, b_1, c_1 ismeretlen paraméterek. Jelölje $p_i = \Phi(x_i)$ közelítendő függvény értékeit valamilyen $\{x_i\}$ adott ponthalmazon, akkor a

$$\min_{a_1,b_1,c_1} \sum_{i=1}^{n} [p_i - (a_1x_i^2 + b_1x_i + c_1)]^2$$

feladatot kell megoldani. Az a_1, b_1, c_1 szerinti differenciálás után a deriváltakat egyenlővé tesszük nullával és az egyenletrendszert megoldjuk az ismeretlen paraméterekre (a megfelelő kifejezések formális leírását lásd például a [De 98a] cikkben).

A második típusú, logaritmikusnak nevezett közelítést keressük $t_2(x) = e^{g_2(x)}$ formában, ahol $g_2(x) = a_2x^2 + b_2x + c_2$. Az ilyen formában felírható közelítés keresését az motiválja, hogy ha $g_2(x)$ közelíti a $\log(\Phi(x))$ függvényt, akkor a $e^{g_2(x)}$ függvény a $\Phi(x)$ függvénynek lesz egy közelítése. Tehát a megoldandó minimalizálási feladat alakja

$$\min_{a_2,b_2,c_2} \sum_{i=1}^{n} [\log(p_i) - (a_2x_i^2 + b_2x_i + c_2)]^2 w_i, \tag{4.3}$$

ahol a w_i súlyokat a $w_i = p_i^2/s$ egyenlőségek definiálják; itt $s = \sum_i p_i^2$, vagyis a súlyokat normalizáljuk. Ezt a súlyozást azért vezettük be, hogy a logaritmikus transzformáció torzító hatását ellensúlyozzuk, amint azt a következőkben kifejtjük.

Ha adott i és j indexekre fennáll $p_i - e^{g_2(x_i)} = p_j - e^{g_2(x_j)}$ (vagyis a hiba ugyanaz az x_i és x_j pontokra, akkor a $g_2(x)$ függvénynek a $\log p$ körüli, konstans és lineáris tagot tartalmazó sorfejtéséből látható, hogy a w_i súlyok fenti megválasztása biztosítja azt, hogy a hibák a logaritmikus transzformáció után is körülbelül ugyanaz lesz: $[\log p_i - g_2(x_i)]w_i$ közel lesz $[\log p_j - g_2(x_j)]w_j$-hez (lásd a 4.2.2 részt, vagy [De 98a]). Jegyezzük meg, hogy mivel $\Phi(x)$ logkonkáv [Pr 95], ezért a $\log \Phi(x)$ függvény konkáv, így akármilyen, többékevésbé jó közelítésének is konkávnak kell lennie, tehát $g_2(x)$ is konkáv lesz.

Természetesen a (4.3) feladat felírható folytonos x esetére is:

$$\min_{a,b,c} \int [\log \Phi(x) - (ax^2 + bx + c)]^2 w(x) dx,$$

amely egy feltétel nélküli minimalizálás és az előzőekhez hasonlóan lehet a megoldását előállítani. Sajnos a $w(x) = [\Phi(x)]^2$ súlyfüggvény használata esetén néhány nehezen kiszámítható értékre is szükségünk lenne. Bár egy $w(x) = const$ állandó súlyfüggvényt használva némely integrált (a standard normális momentumait) közvetlenül meghatározhatjuk, a tárgyalás áttekinthetősége érdekében mégiscsak a (4.3) diszkrét formát fogjuk használni a továbbiakban.

Végül a fordított-logaritmikus elnevezésű közelítést adjuk meg, melynek alakja $t_3(x) = 1 - e^{g_3(x)}$, ahol $g_3(x) = a_3x^2 + b_3x + c_3$ a belső függvény. A $g_3(x)$ a legkisebb négyzetek eljárásának segítségével meghatározott közelítése a $\log(1 - \Phi(x))$ függvénynek, amelyet a fentiek szerint lehet kiszámítani. A t_2 közelítés meghatározásához hasonlóan itt is bevezetjük a $w_i = (1 - p_i)^2/s, s = \sum_i (1 - p_i)^2$ súlyokat, és a

$$\min_{a_3,b_3,c_3} \sum_{i=1}^{n} [\log(1 - p_i) - (a_3x_i^2 + b_3x_i + c_3)]^2 w_i$$

feladatot oldjuk meg az ismeretlen a_3, b_3, c_3 paraméterek meghatározása céljából.

A továbbiakban megvizsgáljuk a $\Phi(x)$ függvény deriváltját és néhány ezzel kapcsolatos függvényt, hogy a $\log(1 - \Phi(x))$ függvény konkávitását megmutassuk (ez egyébként Prékopa logkonkávitási eredményei alapján triviálisan igaz, de itt egy egyszerűbb gondolatmenetet alkalmazunk). Mivel $\Phi(x)$ logkonkáv, a logaritmusának második deriváltja negatív, azaz

$$[\log \Phi(x)]'' = -\frac{\phi(x)}{\Phi(x)} \left[\frac{\phi(x)}{\Phi(x)} + x \right] < 0$$

fennáll. Az utolsó egyenlőtlenségből pedig látható, hogy a

$$\phi(x)/\Phi(x) > -x \tag{4.4}$$

egyenlőtlenség teljesül. A $\log(1 - \Phi(x))$ függvény első deriváltja $-\phi(x)/\Phi(-x)$, amely negatív, a második derivált pedig

$$[\log(1 - \Phi(x))]'' = -\frac{\phi(x)}{\Phi(-x)} \left[\frac{\phi(x)}{\Phi(-x)} - x \right], \tag{4.5}$$

ahol felhasználtuk a $\Phi(x) = 1 - \Phi(-x)$ és a $\phi(x) = \phi(-x)$ egyenlőségeket. Tekintsük most a $[\log(1 - \Phi(x))]''$ függvényt, ez nyilván negatív $x \leq 0$ esetén. Az $x > 0$ eset vizsgálatához helyettesítsük a $\phi(x)/\Phi(-x) \geq \phi(x)/\Phi(x)$ egyenlőtlenséget a (4.5) jobboldalának második tagjába. Ekkor a (4.4) felhasználásával látható, hogy

$$[\log(1 - \Phi(x))]'' < 0, \ x \in R^1 \tag{4.6}$$

igaz, tehát a $\log(1-\Phi(x))$ függvény is konkáv. Ez azt jelenti, hogy a t_3 közelítésben szereplő $g_3(x)$ függvénynek is konkávnak kell lennie minden ésszerű közelítés esetén.

Egy fontos megjegyzést teszünk itt: céljainknak csak a közelítő függvények csonkolt változata felel meg. A $g_2(x)$ és a $g_3(x)$ függvények hasonlóak, – mindkettő konkáv függvény az $a_2 < 0$ illetőleg az $a_3 < 0$ esetén, de egy lényeges szempontból különbözőek. A $g_2(x)$ közelítő függvénynek szigorúan monotonnak növekvőnek kell lennie, mivel $\log\Phi(x)$ szigorúan monoton növő; ebből következik, hogy a $g_2(x)$ függvénynek csak a „baloldala" használható. Definiáljuk a $x_t = -b_2/(2a_2)$ csonkolási pontot és legyen

$$g_2(x) = a_2x^2 + b_2x + c_2, \text{ ha } x \in (-\infty, -b_2/(2a_2)].$$

A $1-\Phi(x)$ illetőleg a $\log(1-\Phi(x))$ függvények monoton csökkennek, így a $g_3(x)$ kvadratikus függvénynek csak a csökkenő része („jobboldala") használható a közelítésben, tehát legyen

$$g_3(x) = a_3x^2 + b_3x + c_3, \text{ ha } x \in [-b_3/(2a_3), \infty)$$

Az eredeti függvénynek a csonkolt része, ha szükséges, helyettesíthető egy állandóval (amelynek értéke $g(x_t) = g(-b/(2a))$, a csonkolási pontban felvett függvényérték).

4.1.2. A normális eloszlásfüggvény közelítései

Az approximáló függvények meghatározását csak az $x > 0$ esetre végezzük el, hiszen az $x < 0$ esetén a $\Phi(-x) = 1 - \Phi(x)$ egyenlőség alkalmazható. Ebben a részben az előzőleg bevezetett t_1, t_2, t_3 jelöléseket használjuk, de ezek csak a függvények általános formáját jelölik, a tényleges paraméterek eltérhetnek.

A fordított-logaritmikus közelítés

A t_3 approximáció általában jó eredményt ad, mivel jól utánozza a normális eloszlás eloszlásfüggvényét, különösen igaz ez az eloszlás farkára. Ezt a t_3 közelítés néhány tulajdonságának megadásával mutatjuk meg.

a.) Az $a_3 < 0$ esetén a $g_3(x) = a_3x^2 + b_3x + c_3$ függvény konkáv, monoton csökkenő $x > x_t = -b_3/(2a_3)$ esetén, amiből a $t_3(x) = 1-\exp(g_3(x))$ monoton növekvő tulajdonsága következik. Vegyük észre, hogy $\lim_{x\to\infty} g_3(x) = -\infty$, tehát $\lim_{x\to\infty} t_3(x) = 1$ is fennáll. Mindkét tulajdonság – a monoton növekvés és az 1-hez való konvergencia – igaz a $\Phi(x)$ eloszlásfüggvényre is.

b.) A $\Phi(x)$ függvény $\phi(x)$ deriváltja mindig pozitív, az $x > 0$ esetén monoton csökkenő és nullához tart $x \to \infty$-re, a konvergenciája $O(e^{-x^2})$. A közelítő t_3 függvény hasonlóan viselkedik, mivel $t_3'(x) = [1 - e^{g_3(x)}]' = -(2a_3x + b_3)e^{g_3}(x)$, amely szintén pozitív $x > x_t$ esetén, továbbá $t_3'(x)$ tart a nullához $O(xe^{-x^2+x})$ sebességgel, amely (nem nagyon nagy x értékekre) majdnem azonos az eloszlásfüggvény deriváltjának viselkedésével. A $\log(1-\Phi(x))$ második deriváltja

$$(\log(1-\Phi(x)))'' = -\frac{\phi(x)}{\Phi(x)}\left[\frac{\phi(x)}{\Phi(x)} - x\right],$$

amely negatív és 0-hoz tart $O(e^{-x^2})$, ha $x \to +\infty$. A $t_3(x)$ függvényre pedig

$$t_3''(x) = -[(2a_3x + b_3)^2 + 2a_3]e^{g_3(x)},$$

ami szintén negatív, ha $2a_3x + b_3 > \sqrt{-2a_3}$, vagyis ha $x > (\sqrt{-2a_3} - b_3)/(2a_3)$ – hiszen a_3, b_3 negatívak a mi esetünkben. Továbbá $\lim_{x\to\infty} t_3''(x) = 0$ és a konvergenciára $O(x^2e^{-x^2})$ fennáll, ami az eloszlásfüggvénynél tapasztaltaknak jól megfelel.

Tegyük fel, hogy az előző szakaszban leírtak szerint akarunk a $[0, r]$ intervallumban egymástól egyenlő távolságra lévő n pontban felvett függvényértékek segítségével egy közelítést meghatározni. Természetesen az n és az r értékeitől függően a a_3, b_3, c_3 paraméterek értékei némi ingadozást fognak mutatni. Mivel a $t_3(x)$ az általános alakja miatt növekvő x esetén jól közelít, ezért az r értékét megpróbáljuk lehetőleg kicsinek tartani – arra kényszerítve az approximáló függvényt, hogy az origóhoz közel is viszonylag jó közelítést adjon.

A továbbiakban megadjuk azt a három approximációt, amelyek a legjobban viselkedtek a numerikus számítások szerint és amelyeket a standard normális eloszlás eloszlásfüggvényének közelítésére ajánlunk.

Egy általános közelítés

Ezt az approximációt javasoljuk azon esetekben, amikor a számítások egyszerűségére van szükségünk és egy tetszőleges x esetén jó közelítést szeretnénk kapni. Az approximáló függvény t_3 alakú, és a benne szereplő konstansok értéke a következő:

$$\begin{aligned} \Phi(x) &\sim t_g(x) = 1 - e^{a_3x^2+b_3x+c_3} \\ a_3 &= -0.379254 \\ b_3 &= -0.770566 \\ c_3 &= -0.694893 \end{aligned} \tag{4.7}$$

Ezeket a numerikus értékeket úgy kaptuk meg, hogy a $[-0.05, 2.2)$ intervallumban vettünk fel $n = 225$ ekvidisztans pontot (a közelítés a $[0, \infty)$ félegyenesen használható).

A közelítés legnagyobb abszolút hibája 0.87×10^{-3}, amelyet az $x = 0$ pontban vesz fel. A hibafüggvény további lokális maximumai (és minimumai) a következő pontokban vannak: $x = 0.25$-ben a hiba $= 0.73 \times 10^{-3}$, az $x = 0.96$ pontban a hiba -0.59×10^{-3}, az $x = 2.13$ esetén a hiba 0.72×10^{-3}, ettől a ponttól kezdve a végtelen felé a hiba monoton csökken. Ezek szerint az approximáció megbízható, legalább három tizedesre pontos eredményt ad és nincs korlát az x értékére (a csonkolási pont $x_t \sim -1.016$).

	t_1	t_2	t_3
a	-0.083668	-0.253309	-0.360682
b	0.429250	0.769021	-0.784058
c	0.497643	-0.690378	-0.694043

4.1. táblázat. Az kombinált közelítés állandói.

Kombinált közelítés

Ha a számításokban valamivel nagyobb bonyolultságot is megengedhetünk, akkor használhatjuk ezt a közelítést – az előző általános közelítésnél tapasztaltnál tizedakkora hibát érhetünk el. A félegyenest némileg önkényesen két részre osztjuk, legyenek ezek a részek $[0, 0.95]$ és $[0.95, \infty)$, és különböző közelítéseket használunk ezen a két részen. Az első intervallumon a $t_{c1}(x)$-szel jelölt approximációt, a $[0.95, \infty)$ félegyenesen pedig a $t_{c2}(x)$ jelű függvényt. A $[0, 0.95]$ intervallumon a három különböző fajtájú approximáció egy lineáris kombinációját használjuk (ezért nevezzük a közelítést kombináltnak):

$$t_{c1}(x) = \frac{1}{2}\left[\frac{(t_1(x) + t_2(x))}{2} + t_3(x)\right], x \in [0, 0.95] \tag{4.8}$$

Ezt a formát arra alapozva alakítottuk ki, hogy a t_1 alakú közelítés és a t_2 alakú közelítés hibafüggvénye ellenkező előjelet mutat majdnem minden esetben, hasonlóképpen ellentétes előjelű a $(t_1(x)+t_2(x))/2$ és a $t_3(x)$ approximációk hibafüggvénye. Míg a t_1 és a t_2 abszolút hibái mintegy 2×10^{-3} nagyságúak, az átlaguk abszolút hibája kisebb a 4×10^{-4} értéknél, a $t_{c1}(x)$ függvénynek a $\Phi(x)$ eloszlásfüggvénytől való abszolút eltérése pedig kisebb 4×10^{-5}-nél. A (4.8)-ban szereplő t_1, t_2, t_3 közelítések állandóit a 4.1 táblázatban adjuk meg.

A kombinált közelítés második, $[0.95, \infty)$ félegyenesen megkonstruált függvénye $t_{c2}(x) = t_3(x), x \in [0.95, \infty)$, melynek paraméterei:

$$a_3 = -0.424478, b_3 = -0.669668, c_3 = -0.746823.$$

Ennek a közelítésnek a legnagyobb abszolút hibája 7×10^{-5}, a hiba pedig az $x = 2.8$ értéknél nagyobb x értékekre csökken. Összefoglalva a kombinált approximálásra használt függvényeinket kapjuk a következőt:

$$t_c(x) = \begin{cases} t_{c1}(x), & \text{ha } x \in [0, 0.95), \\ t_{c2}(x), & \text{ha } x \in [0.95, \infty). \end{cases} \tag{4.9}$$

Dekomponált közelítés

Az előző közelítésnél tett felbontáshoz hasonlóan járunk el: a $[0, \infty)$ félegyenest felbontjuk öt intervallumra és a maradék félegyenesre. A használt dekompozíció némileg önkényes,

i	$[d_i, e_i)$	a	b	c
1	[0.00,0.25)	-0.332731	-0.796469	-0.693163
2	[0.25,0.55)	-0.357923	-0.782874	-0.695015
3	[0.55,0.90)	-0.382191	-0.755848	-0.702595
4	[0.90,1.35)	-0.406612	-0.711912	-0.722454
5	[1.35,2.00)	-0.431045	-0.645689	-0.767481
6	[2.00,∞)	-0.452221	-0.562700	-0.848839

4.2. táblázat. A dekomponált közelítés állandói.

számítógépes kisérletezések után jutottunk el a használt értékekhez. A felbontás minden részében egy $t_3(x)$ alakú approximációt konstruáltunk, jelölje az i-edik részben lévő közelítést $t_{p,i}(x)$. A teljes dekomponált approximáció

$$t_p(x) = t_{p,i}(x), \text{ ha } x \in [d_i, e_i), i = 1, \cdots, 6 \tag{4.10}$$

alakban írható fel, ahol a $t_{p,i}$ függvények értékét a $[d_i, e_i)$ intervallumon kívül nullának tekintjük. A felbontás, a $[d_i, e_i)$ intervallumok és a szereplő $g_3(x)$ állandóit a 4.2 táblázatban adtuk meg. A közelítés abszolút hibája a teljes $[0, \infty)$ félegyenesen kisebb mint 1.2×10^{-5} (a hiba túlnyomórészt 6×10^{-6} körüli).

A numerikus számításokat egy személyi számítógépen végeztük a Microsoft Excel, Version 4.0a program használatával, a standard normális eloszlás eloszlásfüggvényének értékeit pedig az ott található függvénnyel számítottuk ki. Lehetséges, hogy az állandók megadott értékeire egy pontosabb (például az egydimenziós normális eloszlásfüggvény duplapontosságú értékeit előállító) szubrutin felhasználásával némileg eltérő értékeket kapnánk, de itt főleg azt akartuk szemléltetni, hogy a t_1, t_2, t_3 alakú közelítések egyszerűségük ellenére elég jó közelítést adnak.

A $t_3(x)$ alakú közelítésnek a $[d_i, e_i)$ intervallumban három olyan pontja volt, ahol a közelítés pontosan a közelítendő függvény értékét vette fel. A legnagyobb hibák pedig a részintervallumok elején és végén léptek fel. Ezeket a hibákat úgy csökkentettük, hogy az x_i alappontokat nem a $[d_i, e_i)$ intervallumon, hanem a $[d_i - \varepsilon, e_i + \varepsilon)$ intervallumon vettük fel egy $\varepsilon \sim 0.01 - 0.1$ értékkel (természetesen a közelítés csak a $[d_i, e_i)$ intervallumon használható).

A fentebbi elgondolásokat használva (több osztóponttal, vagy másmilyen felbontással) tetszőlegesen további approximáló függvények konstruálhatók, amelyek a teljes félegyenesen, vagy az adott felhasználó által fontosnak tartott intervallumon még jobb közelítést adhatnak.

4.2. A többdimenziós normális eloszlás egy egyenes mentén való közelítései

Az előző részben leírtakhoz hasonlóan egydimenziós közelítő függvényeket konstruálunk az m-dimenziós normális eloszlás eloszlásfüggvényének egy egyenes mentén felvett értékeihez. A többdimenziós normális eloszlás értékeit Monte Carlo módszerrel lehet meghatározni (lásd a 3. fejezetben leírtakat), így a kapott függvényértékek egy véletlen hibával (zajjal) terheltek, tehát a közelítések a valószínűségszámításban használt terminológia szerint az eredeti függvény regressziós becslései lesznek. A becslésekben szereplő paraméterek meghatározását mindig az előző szakaszban használt legkisebb négyzetek módszerével végezzük, vagyis az L_2 minimális normájú becslést használjuk.

4.2.1. A normális eloszlás lineáris regressziós becslései.

Legyen a 0 várható értékű, R korreláció mátrixszú m-dimenziós normális eloszlás eloszlásfüggvénye adott a következőképpen:

$$\Phi(\mathbf{h}) = \int_{-\infty}^{h_1} \cdots \int_{-\infty}^{h_m} (2\pi)^{(-m/2)} |R|^{-1/2} \exp\left\{-\frac{1}{2}\mathbf{z}R^{-1}\mathbf{z}\right\} d\mathbf{z}. \tag{4.11}$$

Feltesszük, hogy az eloszlás nem degenerált, tehát R pozitív definit mátrix, minden komponens szórása 1 – a nemstandard eloszlások lineáris transzformációval kezelhetők.

A $\Phi(\mathbf{h})$ függvényértékek kiszámítására használt Monte Carlo módszerek egy η valószínűségi változó $y_1, \ldots, y_n$ független realizációit állítják elő, ahol $\mathrm{E}(\eta) = \Phi(\mathbf{h})$. A függvényérték

$$y = \frac{1}{n}\sum_{i=1}^{n} y_i$$

torzítatlan becslését használják a numerikus számításokban a $\Phi(\mathbf{h})$ közelítő értékeként. Ha $D^2(\eta) = \sigma^2$, akkor $D^2(y) = \sigma^2/n$, vagyis az y becslés szórása $\sigma/\sqrt{n}$. A centrális határeloszlás tétel alapján az η valószínűség változóról feltesszük, hogy normális eloszlású, azaz $\eta \in N(\Phi(\mathbf{h}), \sigma)$.

Jelölje a többdimenziós normális eloszlás eloszlásfüggvényének egy egyenes mentén való $R^1 \to R^1$ függvényét $f(x)$, vagyis legyen

$$f(x) = \Phi(\mathbf{z} + x(\mathbf{d} - \mathbf{z})),$$

ahol $\mathbf{z}, \mathbf{d}$ rögzítettek, valamint feltesszük, hogy $\mathbf{d}$ növekvő irány, tehát $f(x_1) < f(x_2)$, ha $x_1 < x_2$ – ez a feltevés csak a csonkolás és a gyökkeresés leírásának leegyszerűsítésére szolgál, de egyébként nem lényeges kikötés és nem szűkíti az általánosságot.

Három olyan eljárást írunk le, amelyek segítségével az $f(x)$ függvény közelítéseit lehet megkonstruálni. Ezeknek az egyváltozós közelítéseknek az alkalmazása végsősoron azt eredményezi, hogy az m-dimenziós térben lévő feladatok helyett csak egydimenziós feladatokat kell megoldani (például a gyökkeresésnél, vagy a gradiens kiszámításánál). Továbbá

egy iteratív eljárás segítségével a közelítések egy adott pontban egyre pontosabbakká tehetők – anélkül, hogy pontosan tudnánk, hogy hol van ez a pont.

Tegyük fel, hogy a rendelkezésünkre áll egy Monte Carlo eljárás, amelynek segítségével az $f(x_i), i = 1, \ldots, n$ függvényértékek (zajos) p_i becsléseit meg tudjuk határozni, ahol p_i egy valószínűségi változó, amelyre $p_i = f(x_i) + \xi,\ \xi \in \widetilde{N}(0, \widetilde{\sigma})$. Itt $\widetilde{N}(0, \widetilde{\sigma})$ a nulla várható értékű, σ szórású egydimenziós normális eloszlásnak egy csonkolt változata, vagyis a csonkolt eloszlás 0 várhatóértékű, $\widetilde{\sigma}$ szórású, sűrűségfüggvénye pedig

$$\widetilde{\phi}(x) = C \exp\left\{-\frac{x^2}{2\sigma^2}\right\}, \quad x \in [-3\sigma, 3\sigma],$$

ahol a konstans értéke $C = 1.0028/(\sqrt{2\pi}\sigma)$. Feltesszük még, hogy $0 < f(x_i) - 3\sigma, f(x_i) + 3\sigma < 1$ fennáll, vagyis $0 < p_i < 1, \forall i$.

Legyen adott az x_i pontoknak egy halmaza, amelyekhez meghatározhatjuk a Φ függvényértékek p_i közelítéseit, tehát rendelkezésünkre áll a $S = \{x_i, p_i\}_{i=1}^n$ halmaz. Ebből kiindulva a $f(\cdot)$ függvénynek egy $t(\cdot)$ regressziós becslését határozzuk meg. A $t(\cdot)$ alakjára vonatkozóan három különböző feltevést fogadunk el, ennek megfelelően háromfajta becslést konstruálunk. Itt tulajdonképpen azt használjuk ki, hogy a 4.1 részben alkalmazott közelítések viszonylag jól közelítették a normális eloszlásfüggvényt, ezért a hasonló alakú becslések várhatólag jók lesznek itt is.

Az alapbecslés – az eredeti $f(\cdot)$ függvény kvadratikus approximációja

Keressük a $t_1(x)$ becslést, amelynek alakja $g_1(x) = a_1x^2 + b_1x + c_1$, ahol az a_1, b_1, c_1 paramétereket úgy határozzuk meg, hogy a hibák négyzeteinek összege minimális legyen. Ezt a

$$\min_{a_1,b_1,c_1} \sum_{i=1}^{n} [p_i - (a_1x_i^2 + b_1x_i + c_1)]^2 \tag{4.12}$$

feladat megoldásával lehet elvégezni. Az elsőrendű szükséges optimalitási feltételek alapján a fenti összeget az a_1, b_1, c_1 szerint deriváljuk, és a deriváltakat 0-val egyenlővé tesszük. A keletkezett egyenletrendszer lineáris a paraméterekben, tehát könnyen megoldható (a teljes jelölésrendszer és megoldás például a [De 98b] cikkben található).

Nyilvánvalóan a $0 \le f(x) \le 1$ egyenlőtlenségek teljesülnek, tehát azonnal csonkolnunk kell a kapott függvényt, ezt null–csonkolásnak nevezzük: jelölje I a számunkra elfogadható intervallumot $I = \{x | 0 \le g_1(x) \le 1\}$, és ekkor becslésünk alakja $t_1(x) = g_1(x)$, ha $x \in I$.

A feltevés alapján $f(x)$ monoton növekvő, tehát a $g_1(x)$ függvénynek a monoton növekvő része használható, mint az $f(x)$ függvény egy becslése. A Φ eloszlásfüggvény logaritmikusan konkáv, tehát a $g_1(x)$ függvénynek is konkávnak kell lennie (most eltekintünk azoktól az esetektől, amikor p_i kiszámított értékekben fellépő véletlen hibák miatt a $g_1(x)$ függvény esetleg konvexszé válik). Ennek alapján egy újabb csonkolást hajtunk végre, amelyet az állandó–csonkolásnak nevezünk. Legyen a $g_1'(x_t) = 0$ egyenlet megoldása

$x_t = -b_1/(2a_1)$, akkor a null–csonkolás és az állandó–csonkolás figyelembevételével a $t_1(x)$ becslést a következőképpen definiáljuk:

$$t_1(x) = \begin{cases} \max(0, \min(1, g_1(x))), & \text{ha } -\infty < x \le x_t, \\ \max(0, \min(1, g_1(x_t))), & \text{ha } x_t < x < \infty, \end{cases}$$

ahol $g_1(x_t) = c_1 - b_1^2/(4a_1)$. Egyes esetekben másmilyen csonkolásra is szükségünk lehet; nevezzük lineáris csonkolásnak azt, amikor a függvény levágott részét a $g_1(x_t)$ állandó helyett egy olyan monoton növő lineáris függvénnyel helyettesítjük, amely a $g_1(x)$ függvény érintője az $x_{tt} = x_t - \kappa$ pontban (ahol $\kappa > 0$ egy kis szám), de csak addig, míg az 1 értéket el nem éri. Pontosabban legyen az egyenes egyenlete $a_t x + b_t$, akkor a

$$a_t x_{tt} + b_t = g(x_{tt}),\ 2a_1 x_{tt} + b_1 = a_t$$

egyenlőségekből határozhatók meg az egyenes a_t, b_t paraméterei. Tehát a $t_1(x)$ alapbecslés a null-csonkolás és a lineáris csonkolás esetén

$$t_1(x) = \begin{cases} \max(0, \min(1, g_1(x))), & \text{ha } -\infty < x \le x_{tt} \\ \max(0, \min(1, a_t x + b_t)), & \text{ha } x_{tt} < x \le (1 - b_t)/a_t, \\ 1, & \text{ha } (1 - b_t)/a_t < x < \infty. \end{cases}$$

A további becslések esetén az egyszerűség kedvéért csak az állandó–csonkolást írjuk le, bár a null–csonkolást mindig el kell végezni és a lineáris csonkolás is végrehajtható (az állandó csonkolás helyett). Hasonlóképpen nem foglalkozunk azzal az esettel, amikor $f(x)$ függvény definíciójában szereplő $\mathbf{d}$ irány nem növekvő irány. Ezek az elhanyagolások csak a gyökkeresés folyamatának egyszerűbb leírásához vezetnek – az $f(x)$ mindig közelíthető egy $g_1(x)$ alakú függvénnyel.

Jegyezzük meg, hogy a $t_1(x)$ approximáló függvény monoton növővé tehető, de a konkávitását nem lehet garantálni. Hasonlóképpen a további $t_2(x), t_3(x)$ becslések esetén sem lehet a konkávitást biztosítani, csak arra hivatkozhatunk, hogy a legtöbb gyakorlati számítás során ezt aszimptotikusan (a felvett pontok számának megfelelően naggyá választásával) mindig elértük.

Logaritmikus becslés – a $\log f(\cdot)$ kvadratikus approximációja

Az $f(x)$ függvény közvetlen approximációja helyett most egy $g_2(x) = a_2x^2 + b_2x + c_2$ alakú függvényt használunk a $\log f(x)$ függvény közelítésére, tehát az $f(x)$ közelítésére használt logaritmikus becslésünk $t_2(x) = \exp\{g_2(x)\}$. Ez a becslés is könnyen meghatározható: egy lineáris egyenletrendszerből kaphatók a paraméterek. Vezessük be a $q_i = \log p_i$ jelölést, akkor a megoldandó feladatunk

$$\min_{a_2,b_2,c_2} \sum_{i=1}^{n} \left[q_i - (a_2x_i^2 + b_2x_i + c_2)\right]^2 w_i, \tag{4.13}$$

ahol a w_i normalizált súlyokat is használtuk (ezeket később adjuk meg). Az a_2, b_2, c_2 paramétereket a (4.13) feladatból az előzőleg leírtak szerint lehet kiszámítani (lásd még [De 98b]). Prékopa eredményei alapján [Pr 95] a $\Phi(\mathbf{h})$ logkonkáv, tehát a $\log f(x)$ függvény konkáv és monoton növekvő, tehát $g_2(x)$ függvényt is konkávvá és monoton növővé kell tenni. Legyen $x_t = -b_2/(2a_2)$ a csonkolási pont, amivel a $t_2(x)$ becslés a következőképpen adható meg.

$$t_2(x) = \begin{cases} \max(0, \min(1, \exp\{g_2(x)\})), & \text{ha } -\infty < x \leq x_t, \\ \max(0, \min(1, \exp\{g_2(x_t)\})), & \text{ha } x_t < x < \infty. \end{cases}$$

A fordított–logaritmikus becslés – a $\log(1 - f(x))$ kvadratikus approximációja

A $\log(1 - f(x))$ függvényt egy $g_3(x) = a_3x^2 + b_3x + c_3$ alakú függvénnyel közelítjük, tehát a $t_3(x) = 1 - \exp\{g_3(x)\}$ függvény az eredeti $f(x)$ függvény egy becslése lesz. Ez is egy lineáris regresszió, melynek paramétereit a

$$\min_{a_3,b_3,c_3} \sum_{i=1}^{n} [r_i - (a_3x_i^2 + b_3x_i + c_3)]^2 w_i.$$

minimalizálási feladat elsőrendű szükséges feltételeiből lehet meghatározni – itt használtuk az $r_i = \log(1 - p_i), i = 1, \ldots, n$ jelölést, a w_i normalizált súlyokat később határozzuk meg. Az a_3, b_3, c_3 meghatározása egyszerűen úgy végezhető, hogy az a_2, b_2, c_2 esetén kapott kifejezésekben a q_i jelöléseket r_i-re cseréljük. A $\log(1-f(x))$ függvény monoton csökkenő. Egy megfelő csonkolás eléréséhez feltesszük, hogy az approximáló függvény konkáv (a konvex eset hasonlóan kezelhető) és az $x_t = -b_3/(2a_3)$ pontot használjuk csonkolási pontként. A fordított logaritmikus becslés a null– és az állandó–csonkolás alkalmazása esetén a következő:

$$t_3(x) = \begin{cases} \max(0, \min(1, 1 - \exp\{g_3(x_t)\})), & \text{ha } -\infty < x \leq x_t \\ \max(0, \min(1, 1 - \exp\{g_3(x)\})), & \text{ha } x_t < x < \infty. \end{cases}$$

Emlékeztetünk arra, hogy a 4.1 részben leírtak szerint egy ilyen alakú és módon számolt, az egydimenziós normális eloszlásfüggvényt közelítő függvény három tizedesre pontos eredményt ad, így remélhetőleg ez a becslés jól közelíti az m-dimenziós normális eloszlásfüggvény egy egyenes mentén való függvényét is.

4.2.2. A logaritmikus transzformáció mellékhatásai

Az $f(x)$ függvényérték hibája (a zaj) a szokásos feltevések alapján nulla várható értékű, és azonos szórású minden x esetén. Az előző szakaszokban leírt logaritmikus és fordított–logaritmikus becsléseiben szereplő logaritmikus transzformáció megváltoztatja a hiba eloszlását, így a várható értékét és a szórását is. Pontosabban míg a $p = f(x_0) + \xi$ zajos függvényértékben $Ep = f(x_0) + E\xi = f(x_0)$ és $Dp = D\xi = \sigma$ minden x_0 esetén, addig

$\log p = \log [f(x_0) + \xi]$ várható értéke nem $\log f(x_0)$ és szórása is függ x_0-tól. Szeretnénk elérni, hogy a transzformáció után kapott hiba is nulla várható értékű és azonos szórású legyen, vagy ezt legalábbis közelítőleg szeretnénk biztosítani.

A szórásban bekövetkezett változást súlyok bevezetésével lehet csökkenteni, míg a várható érték megváltozását egy korrekciós, javító tag hozzáadásával lehet ellensúlyozni. A súlyozás használatát minden, a logaritmikus transzformációt használó becslés esetében javasoljuk, ha az x_i alappontok egy széles tartományban szóródnak (ha az $f(x_i)$ függvényértékek különbsége nagyobb, mint egy adott érték, például σ – ugyanis az x_i pontok tartományát nem lehet jól jellemezni az $|x_i - x_j|^2$ eltéréssel, ezért inkább az $|f(x_i) - f(x_j)|$ függvényértékek közti különbséget használjuk). A korrekciós tag használatát csak akkor javasoljuk, ha nagyon pontos függvényértékekre van szükségünk.

Súlyok a logaritmikus transzformáció esetén

Az egyszerűség kedvéért csak a $t_2(x)$ becslés vizsgálatával foglalkozunk, de a gondolatmenet és a megadott képletek nehézség nélkül átvihetők a fordított–logaritmikus, vagy akármilyen más, a logaritmikus transzformációt használó becslésre.

Tegyük fel, hogy az x_0 helyen egy Monte Carlo módszerrel a p_0 értéket határoztuk meg, $p_0 \in \widetilde{N}(f(x_0), \widetilde{\sigma})$, azaz $p_0 = f(x_0) + \xi$ írható egy $\xi \in \widetilde{N}(0, \widetilde{\sigma})$ valószínűségi változó segítségével. Természetesen a tényleges számításokban null–csonkolást hajtunk végre, vagyis értékünk a $p_0 = \max(0, \min(1, f(x_0) + \xi))$ lesz, de a leírás egyszerűségének érdekében a p_0 normális eloszlását tesszük fel (ami numerikus szempontból megengedhető helyettesítés egy kis környezetben).

Tegyük fel, hogy a közelítésünk teljesen pontos, vagyis $g(x) = f(x)$. Ekkor az alapbecslés esetén a $[p_0 - f(x_0)]^2 = \xi_i^2$ hibanégyzeteket összegezzük és minimalizáljuk, ahol a ξ_i hibák függetlenek az $f(x_0)$ függvényértéktől és egymástól. A $t_2(x)$ becslés esetén a minimalizálandó összegben szereplő tagok $[\log p_0 - \log f(x_0)]^2$ alakúak, amelyek függenek az $f(x_0)$ értéktől. Fejtsük ki a logaritmikus függvényt, a $\log p_0 = \log(f(x_0) + \xi)$ értéket az $f(x_0)$ körül, akkor azt kapjuk, hogy

$$\log p_0 = \log f(x_0) + \frac{1}{f(\overline{x})}\xi, \tag{4.14}$$

ahol $f(\overline{x}) \in [f(x_0), f(x_0) + \xi]$ a középértéktétel alapján. Átrendezve a (4.14) tagjait kapjuk a

$$[\log p_0 - \log f(x_0)] \cdot f(\overline{x}) = \xi$$

egyenlőséget. Ezek szerint a minimalizálandó összegben szereplő tagok függetlenné tehetők az $f(x_0)$-tól, ha $f^2(\overline{x})$ értékű súlyokat teszünk az egyes $[\log p_0 - \log f(x_0)]^2$ tagokhoz. Bár az $f(\overline{x})$ értékét nem ismerjük, de ez közelíthető a $p_0 = f(x_0) + \xi$ és az $f(x_0)$ átlagával – ezekből az első ismert, a második ismeretlen, de viszont közelíthető valamelyik előzőleg leírt becslés segítségével. Tehát a w_i súlyokat a következő egyenletek adják:

$$\begin{aligned} w_i &= w_i^*/s, \\ w_i^* &= [p_i + t^*(x_i)]^2/4,\ i = 1, \ldots, n, \\ s &= \sum_{i=1}^{n} w_i^*, \end{aligned}$$

ahol a $t^*(\cdot)$ becslés akár a kiszámítandó becslés egy korábbi változata, akár az $f(x)$ valamelyik másmilyen, előzőleg meghatározott becslése lehet. A $t^*(\cdot)$ szerepében használhatjuk a $t_0(x) = a_0x+b_0$ lineáris regressziós becslést is, amely igen egyszerűen meghatározható.

A fordított-logaritmikus becslésekben használt súlyokat szintén meg lehet határozni az előző érvelés segítségével, és ezekre

$$\begin{aligned} w_i &= w_i^*/s, \\ w_i^* &= [1 - p_i + 1 - t^*(x_i)]^2/4, i = 1, \ldots, n, \\ s &= \sum_{i=1}^{n} w_i^*. \end{aligned}$$

Amint ezt megjegyeztük, a logaritmikus transzformáció által a hiba szórásnégyzetében okozott változás csak abban az esetben lesz zavaró a numerikus számításainkban, ha az $f(x_i)$ értékek nagyon különbözőek. Ezt a „nagyon különbözőséget" a következő módon lehet kvantifikálni. Tegyük fel, hogy kiszámítottuk az első $t^*(x)$ közelítést, ha most a $t^*(\max_i x_i) - t^*(\min_i x_i) < \sigma$ egyenlőtlenség fennáll, akkor gyakorlatilag nincs szükség a súlyok kiszámítására és használatára, mert a függvénymeghatározásban tapasztalt σ hiba értéke nagyobb a logaritmikus transzformáció által okozott hibánál, így ez utóbbi elhanyagolható.

A becslés korrekciója

Eredetileg a mintavételi hiba 0 várható értékű – ha egy korrekciós tagot hozzáadunk a becsléshez, akkor a transzformált hiba várható értékét közel 0 értékűvé tudjuk tenni.

Tegyük fel, hogy adott x-re az $f(x)$ függvényérték egy p közelítő értékét kiszámítottuk, tehát $p = f(x) + \xi, \xi \in \widetilde{N}(0, \widetilde{\sigma})$. A sztochasztikus programozásban a szokásos megbízhatósági szint $f(x) = \mu \sim 0.9$, amely egy $\sigma = 0.05$ hiba esetén nagyon „féloldalas" eloszlású $\log p$ valószínűségi változót eredményez és az ebből eredő eltérés sokkal nagyobb, mint az a korrekció, amelyet a továbbiakban leírunk. Másrészt, egy $\sigma = 0.01$ szórás esetén a probléma gyakorlatilag nem jelentkezik $f(x) \leq 0.95$ függvényértékekre.

Feltesszük, hogy a p várható értéke $\mu = f(x)$, szórása $\widetilde{\sigma}$ (amely nagyon közeli a σ értékhez), sűrűségfüggvénye $\widetilde{\phi}(z) = C \exp\{-((z-\mu)^2)/(2\sigma^2)\}, z \in [\mu - 3\sigma, \mu + 3\sigma]$ egy $C = 1.0028/(\sqrt{2\pi}\sigma)$ állandóval.

Sorbafejtjük $\log p = \log(\mu + \xi)$ függvényt a μ körül, akkor

$$\log p = \log \mu + \frac{1}{\mu}(p-\mu) - \frac{1}{\mu^2}\frac{(p-\mu)^2}{2} \cdots (-1)^{k-1}\frac{1}{\mu^k}\frac{(p-\mu)^k}{k} \cdots . \tag{4.15}$$

A $\log p$ várható értékét úgy kaphatjuk meg, hogy a fentebbi kifejezést használjuk a várható értékben:

$$\mathrm{E}(\log p) = \int_{\mu-3\sigma}^{\mu+3\sigma} C \log z \exp\left\{-\frac{(z-\mu)^2}{2\sigma^2}\right\} dz = \log \mu - \frac{\widetilde{\sigma}^2}{2\mu^2} - \cdots - \frac{\widetilde{\sigma}^{2k} M_{2k}}{(2k)\mu^{2k}} \cdots , \tag{4.16}$$

ahol M_{2k} a $2k$-adik momentuma a $\widetilde{N}$ eloszlásnak, mivel a páratlanadik momentumok nullával egyenlőek. Legyen r egy páros szám, akkor a standard normális eloszlás r-edik momentuma $2^{r/2}\Gamma((r+1)/2)/\sqrt{\pi}$, amely egy divergens összeget adna az $E(\log p)$ várható értékre, de a $\widetilde{N}$ csonkolt normális eloszlásnak korlátos momentumai vannak. Ez a következőképpen látható be. Vegyük a várható értékét a (4.15) összeg $(r+1)$-edik tagjának és helyettesítsük a $\widetilde{\sigma}$ értéket a σ-val, $C = 1.0028/(\sqrt{2\pi}\sigma)$ értéket a $1/(\sqrt{2\pi}\sigma)$ értékkel. Ekkor

$$I_{r+1} = \int_{\mu-3\sigma}^{\mu+3\sigma} C\frac{1}{r\mu^r}(z-\mu)^r \exp\left\{-\frac{(z-\mu)^2}{2\sigma^2}\right\} dz \sim \frac{\sigma^r}{r\mu^r\sqrt{2\pi}} \int_{-3}^{3} y^r \exp\left\{-\frac{y^2}{2}\right\} dy.$$

Ennek az integrálnak egy felső korlátját úgy kaphatjuk meg, hogy az exponenciális függvényt 1-el helyettesítjük, miáltal

$$\begin{aligned} I_{r+1} &\sim \frac{2\sigma^r}{r\mu^r\sqrt{2\pi}} \int_0^3 y^r \exp\left\{-\frac{y^2}{2}\right\} dy \leq \\ &\leq \left(\frac{\sigma}{\mu}\right)^r \frac{2}{r\sqrt{2\pi}} \int_0^3 y^3 dy < \left(\frac{\sigma}{\mu}\right)^r \frac{3^r}{r(r+1)} < \left(\frac{3\sigma}{\mu}\right)^r \frac{1}{r^2}. \end{aligned}$$

Növekvő r esetén ez a felső korlát csökken és számítástechnikai szempontból elhanyagolható értékű, ha $3\sigma < L\mu$ fennáll valamilyen elég nagy L esetén – a legtöbb gyakorlatban felmerülő feladatban $L \sim 0.1$ esetén ez fennáll, így az integrál értékét a L^r/r^2 geometriai sor felülről korlátozza. (Azokban az esetekben, amikor $3\sigma > \mu$, a fordított logaritmikus becslés alkalmazható.) Tehát a (4.16) egyenlet jobboldali összegéből, mivel a tagok gyorsan csökkennek, csak az első két tagot tartjuk meg, miáltal a következő közelítő egyenlőséget kaptuk a $\log p$ várható értékére:

$$\mathrm{E}(\log p) \sim \log \mu - \frac{\sigma^2}{2\mu^2},$$

ahol a jobboldalon szereplő második tagot nevezzük a korrekciós tagnak. Alapjában véve ez a közelítő formula azt jelenti, hogy bár az $f(x)$-nek a $\frac{1}{n}\sum_i f(x_i)+\xi_i$ egy torzítatlan becslése, de $\log(f(x)+\xi)$ értéknek nem torzítatlan becslése a $\frac{1}{n}\sum_i \log[f(x_i)+\xi_i]$

(általában a keresettnél kisebb értékeket kapunk az alkalmazásával), és a kettő közti különbség jól közelíthető a $\sigma^2/(2\mu^2)$ korrekciós taggal. Ez a korrekció egész kicsi, és csak azokban az esetekben használandó, amikor a kiszámítandó becslés értékének megkívánt pontossága közel van a korrekciós tag értékéhez. Mindenesetre, ennek a korrekciónak a használatát csak a számítások legvégső fázisában javasoljuk.

A fordított-logaritmikus becslés esetében a korrekciós tagot a fentiekhez hasonlóan lehet meghatározni, a korrekciós tag képlete ekkor $\sigma^2/(1-\mu)^2$.

4.3. A becslések alkalmazása numerikus feladatokban

A továbbiakban azzal foglalkozunk, hogyan lehet a becsléseinket a numerikus számítások meggyorsítására felhasználni. A sztochasztikus programozás valószínűséggel korlátozott feladata numerikus optimalizálása során felmerülő két részfeladat megoldását részletezzük, a gyökkereső eljárást és a gradiens kiszámítását.

4.3.1. Gyökkereső algoritmus

Legyen p_{rel} adott megbízhatósági szint és tekintsük az

$$f(\overline{x}_r) = p_{rel}$$

egyenletet, melynek pontos gyöke $\overline{x}_r$ – ez általánosan egy kvantilis meghatározási probléma. Feladatunk egy x_r közelítő gyök kiszámítása – ezt röviden gyökkeresési feladatnak nevezzük.

A leírásra kerülő heurisztikus algoritmus a t regressziós becslést használja közelítő gyök meghatározására, ahol t az előzőleg megadott $t_1(x), t_2(x), t_3(x)$ becslések akármelyike lehet. Az eljárás működésének egy magyarázata a következő lehet. Ha a közelítő t függvény az f függvényhez közel van, akkor a $t(x) = p_{rel}$-ből meghatározható közelítő gyök is közel lesz a pontos gyökhöz. Az új x_i mintapontokat a közelítő gyök környékén vesszük fel, így a következő approximáló függvénynek remélhetőleg kisebb lesz a hibája az igazi gyök körül, tehát a következő közelítő gyök is közelebb kerül az igazi gyökhöz.

A továbbiakban egy iteratív eljárás segítségével keresünk egy, a pontos $\overline{x}_r$ gyökhöz tartó $\{x_r\}$ közelítés–sorozatát; a közelítő gyök meghatározásához használt $t(x)$ becslést fokozatosan egyre pontosabbá tesszük a közelítő gyök környékén. Ezt azáltal érjük el, hogy egy összehúzódó intervallum-sorozatot konstruálunk – ezeket a p értéket tartalmazó, értékkészletekre vonatkozó $[\alpha_i, \beta_i]$ intervallumokat konfidencia-intervallumoknak nevezzük. A megfelelő, x értékeket tartalmazó $[a_i, b_i]$ intervallumok a $t^i(x_r) = p_{rel}$ egyenlőségből számítható x_r közelítő gyököket tartalmazzák, ahol $t^i(a_i) = \alpha_i, t^i(b_i) = \beta_i$. Az intervallumok hosszát minden iterációs lépésben csökkentjük egy $\varrho < 1$ redukciós faktorral. Az algoritmus egyes lépéseire vonatkozó részletes megfontolásokat a [De 98a] cikkben közöltük, itt csak az algoritmusok végleges formáját közöljük.

A jelölések egyszerűsítése céljából az algoritmust csak a $t_1(x) = g_1(x) = a_1x^2+b_1x+c_1$ becslésre írjuk le (a szükséges apró módosítások elvégzése után az eljárás a t_2 és a t_3 becslésekre is alkalmazható), valamint a t_1, g_1, a_1, b_1, c_1 mennyiségek indexeit is elhagyjuk. A redukciós faktor értékét $\varrho = 0.6$-nek, az egyes iterációkban használt pontok számát $K = 10$-nek vettük. A kezdeti $[a_0, b_0]$ intervallumot a tapasztalatok szerint szimmetrikusan kellene felvenni a $\overline{x}_r$ gyök körül, de mivel ezt nem ismerjük, egy nagyjából jó becslés elegendő itt. Az algoritmusban ezt a szimmetrizálást közelítőleg úgy érjük el, hogy a jobboldali és a baloldali deriváltak közelítő értékeinek o hányadosát használjuk az algoritmus 2. lépésében.

Gyökkereső eljárás - az $f(x) = p_{rel}$ egyenletre

0. [Előkészítés.] Tegyük fel, hogy a kezdeti $[a_0, b_0]$ intervallum, az ebben lévő $x_{0j}, j = 1, 2, \ldots, K$ pont, az ezekben felvett $p_{0j} \sim f(x_{0j})$ zajos függvényértékek adottak, valamint meghatároztuk már ezek alapján a $g(x)$ becslést. Legyen kezdetben $\sigma_0 = \sigma/\varrho$, az iterációs számláló $i = 0$, és a lépéshossz $stl = (b_0 - a_0)/10$.
1. [A konfidencia-intervallum meghatározása.] Növeljük meg az iterációs számlálót $i = i + 1$, számítsuk ki a $\sigma_i = \varrho\sigma_{i-1}$ értéket és az $x_t = -b/(2a)$ csonkolási pontot.
2. [Az előzetes $[a_i, b_i]$ intervallum kiszámítása.] Határozzuk meg a $g(x_r) = p_{rel}$ egyenletből az x_r közelítő gyököt. Ha $a < 0$, akkor elfogadjuk azt a gyököt, amely kisebb x_t-nél (ha nincs megoldás, akkor legyen $x_r = x_t$). Ha $a > 0$, akkor elfogadjuk az x_t-nél nagyobb gyököt (ha nincs megoldás, akkor legyen $x_r = x_t$). Számítsuk ki a $\delta = 2{*}stl$ értéket és legyen $x^- = x_r-\delta, x^+ = x_r+\delta$. A jobb és baloldali deriváltak közelítő arányát az $o = (p_{rel} - g(x^-))/(g(x^+) - p_{rel})$ hányados mutatja. Legyen $\alpha_i = \max(p_{rel} - o \cdot \sigma_i, 0.0001), \beta_i = \min(p_{rel} + \sigma_i, 0.9999)$.
3. [Az $[a_i, b_i]$ intervallum meghatározása.] Számítsuk ki az $a_i = g^{-1}(\alpha_i)$, $b_i = g^{-1}(\beta_i)$ gyököket. Ha $a < 0$, akkor válasszuk ki azokat a gyököket, amelyek x_t-nél kisebbek, ha $a > 0$, akkor válasszuk ki azokat a gyököket, amelyek x_t-nél nagyobbak (ha nincs megoldás, akkor legyen $a_i = x_t, b_i = a_i + 0.01$).
4. [A $g(x)$ újraszámítása.] Válasszunk ki $K = 10$ darab új pontot az $[a_i, b_i]$ intervallumban: $x_{ij} = (b_i - a_i)(j - 1)/(K - 1) + a_i, j = 1, \ldots, K$ és egy Monte Carlo eljárással határozzuk meg a $p_{ij} \sim f(x_{ij})$, $E(p_{ij}) = f(x_{ij})$ zajos függvényértékeket. Az új pontok halmaza legyen $S_i = \{x_{ij}, p_{ij}\}_{j=1}^K$, és $S = \cup_{l=0}^i S_l$. Határozzuk meg az új $g(x)$ approximációt az összes S-beli pont és függvényérték segítségével. Határozzuk meg az új közelítő $x_r^* = g^{-1}(p_{rel})$ gyököt (ha $a < 0$, akkor vegyük a kisebbet, egyébként pedig a nagyobb értékű gyököt, illetőleg legyen $x_r^* = x_t$, ha nincs megoldás).

5. [A konvergencia ellenőrzése.] Ha $\sigma/\sqrt{(i+1)K} < \varepsilon_0$ fennáll, ahol ε_0 egy előírt hibatűrés, akkor STOP. Egyébként legyen $x_r = x_r^*, stl = 2(b_i - a_i)/K$ és menjünk vissza az 1. lépésre.

Néhány általános megjegyzést teszünk. Számítástechnikai szempontból az algoritmusnak a következő két előnye látható:

(i) a számítógépes eredmények szerint [De 98a] az approximáció hibája az $S = \{x_{ij}, p_{ij}\}$ halmazban megadott pontok számának négyzetgyökével arányosan csökken, vagyis $|f(\overline{x}_r) - f(x_r)| \leq \sigma/\sqrt{(N+1)K}$, ahol N az iterációk száma (ezt a tulajdonságot az algoritmus 5. lépésében használtuk ki).

(ii) a $g(x)$ becslésnek a 4. lépésben leírt újraszámolását nagyon egyszerűen lehet elvégezni, csak az eddig már meghatározott átlagokhoz kell hozzáadni az S_i-beli új pontok megfelelő hatványait (momentumait), vagyis nincs szükség arra, hogy minden iterációban teljesen elölről kezdjük a számításokat. Tehát a regressziós becslés meghatározásával kapcsolatos munka csak lineárisan függ az S-beli pontok (függvénykiszámítások) számától.

Ha a p_{rel} megbízhatósági szint 0.95-0.99 körül van, akkor a $g(x)$ becslés lineáris csonkolását kell használni, mivel a becslés konstrukciója miatt a közelítő gyök nagyon közel lesz a csonkolási ponthoz, és a valódi gyök gyakran a csonkolási pontnál nagyobb.

Az 5. lépésben szereplő megállási kritérium helyett másmilyen megállási elvet is használhatunk; a legegyszerűbb az elvégzendő iterációk N számát előre lerögzíteni. Egy másik lehetőség lehet az, ha mintavétel hibáját számítjuk az egész algoritmus folyamán és a kívánt hibakorlát elérése esetén állítjuk le az algoritmust.

Eljárhatnánk úgy is a gyökkeresés algoritmusában, hogy a teljes m-dimenziós Φ függvényre adunk meg egy m-dimenziós regressziós becslést, és ennek az egy egyenes menti (sokaságon felvett) egydimenziós függvényét használjuk a fenti eljárásban, de a szükséges munka mennyiségét nem indokolja meg a pontosságban elérhető növekedés, ezért ezzel nem foglalkozunk részletesen.

Néhány számítási eredményt közlünk annak megmutatására, hogyan működik a gyökkereső algoritmus rögzített N iterációszám esetén (további számítási eredmények találhatók a [De 98a] cikkben). A számítógépes futásokban a következőképpen állítottuk be az algoritmus állandóit: $K = 10, N = 9, \sigma = 0.05$, vagyis minden eljárásban 100 függvényérték kiszámítását végeztük el és a végső eredmény hibája a legtöbb esetben $\sigma/10 = 0.005$ körül volt. Az első példa esetében egy $m = 2$-dimenziós normális eloszlást használtunk, amelynek komponensei korreláltak voltak. A 3. példában pedig egy tíz-dimenziós normális eloszlást használtunk, amely komponensei páronként korreláltak voltak (az első és a második komponensek egymással korreláltak voltak, de függetlenek az összes többi komponenstől, a harmadik és a negyedik komponensek korreláltak voltak, de függetlenek a többitől, stb.) A 4. példában minden komponens független volt, a kapott numerikus eredményeket a 4.3 táblázatban közöljük.

Példa száma	Dimenzió $m=$	Megb. $p_{rel}=$	"Pontos" gyök $\overline{x}_r$	Becslés	Gyök x_r	Hiba $\lvert f(x_r)-f(\overline{x}_r)\rvert$
1.	2	0.80	1.1114	alap	1.099	0.0025
				log	1.100	0.0035
				f-log	1.131	0.0048
3.	10	0.95	5.895	alap	5.724	0.0024
				log	5.827	0.0010
				f-log	5.999	0.0012
4.	50	0.90	4.276	alap	4.201	0.0038
				log	4.164	0.0050
				f-log	4.216	0.0029

4.3. táblázat. A gyökkeresés eredményei.

4.3.2. Az eloszlásfüggvény gradiense

Prékopa a következő kifejezést ajánlotta a többdimenziós normális eloszlás eloszlásfüggvénye gradiensének kiszámítására [PGDP 76]:

$$\nabla\Phi(\mathbf{h}) = \left\{\widetilde{\Phi}_i(h_1,\cdots,h_{i-1},h_{i+1},\cdots,h_m|h_i)\cdot\varphi(h_i)\right\}_{i=1}^{m},$$

ahol a jobboldalon szereplő $\widetilde{\Phi}_i$ egy $(m-1)$-dimenziós feltételes eloszlásfüggvény, melynek paramétereit (beleértve a korrelációs mátrixát) az eredeti R és $\mathbf{h}$ értékekből meg lehet határozni. A képlet jelentősége az, hogy m darab $(m-1)$-dimenziós eloszlásfüggvény értékének meghatározása árán ki lehet számítani a gradiens értékét.

A gradiens kiszámításának egy másik módja a $\left(f(x+\delta)-f(x-\delta)\right)/(2\delta)$ különbség használatával adódik; de ha egy σ hibájú Monte Carlo módszert használunk az $f(\cdot)$ értékeinek meghatározására, akkor az eredményül kapott közelítés nagy ingadozásokat fog mutatni, tehát ez az eljárás nem vezethet gyakorlatilag használható közelítésre.

A $\nabla\Phi(\mathbf{h})$ értékének regressziós közelítések segítségével való kiszámítására két mód is alkalmazható. Az elsőt komponensenkénti eljárásnak, a másodikat m-dimenziós közelítésnek nevezzük. A komponensenkénti megközelítésben definiáljuk az $f_i(x)=\Phi(\mathbf{h}+x\mathbf{e}_i), i=1,\ldots,m$ függvényeket, ahol $\mathbf{e}_i$ az i-edik egységvektor, kiszámítjuk az n darab $f_i(x_{ij}), j=1,\ldots,n$ zajos függvényértéket azon $\mathbf{h}$ pont egy környezetében, ahol a gradiens értékét meg akarjuk határozni, kiszámítjuk a megfelelő statisztikai becslést minden koordinátatengely mentén és ennek az analitikus deriváltját vesszük a gradiens megfelelő kompo-

$[a,b]$	$f_1(a)$	$f_1(b)$	lineáris	alap	logar.	f-log.
$[-0.8, 0.8]$	0.85	0.98	0.091	0.118	0.121	0.075
$[-0.5, 0.5]$	0.88	0.97	0.097	0.097	0.099	0.076
$[-0.3, 0.3]$	0.91	0.97	0.091	0.090	0.090	0.068
$[-0.45, 0.75]$	0.89	0.98	0.062	0.088	0.088	0.071

4.4. táblázat. Gradiens első komponensének becslése, 4. példa, változó $[a,b]$ intervallumra.

lineáris	alap	logaritmikus	ford.-logar.
0.202	0.202	0.200	0.198
0.252	0.251	0.252	0.253
0.236	0.236	0.246	0.205

4.5. táblázat. Gradiens első komponensének becslése, 1. példa, 3 különböző futás

nense közelítésének. Az egydimenziós becslések deriváltjaira

$$\begin{aligned} t_1'(x) &= 2a_1x + b_1, \\ t_2'(x) &= [2a_2x + b_2]\exp\{a_2x^2 + b_2x + c_2\}, \\ t_3'(x) &= -[2a_3x + b_3]\exp\{a_3x^2 + b_3x + c_3\}. \end{aligned}$$

A második megközelítésben az eloszlásfüggvénynek egy m-dimenziós becsült közelítését használjuk, és a regressziós becslés analitikus gradiensének értékével becsüljük az eredeti gradienst (a [De 98b] cikkben megadtuk a szükséges képleteket).

A komponensenkénti eljárás számítógépes tapasztalatai szerint akkor kapunk általában jó közelítéseket a gradiens értékére, ha az $[a,b]$ intervallumra, amelyben az $f_i(x)$ függvényértékeket kiszámítjuk, teljesülnek a következők:

(i) az intervallum aszimetrikusan helyezkedik el $\mathbf{h}$ körül; a pontoknak mintegy harmada kisebb függvényértéket állítson elő, mint a $\mathbf{h}$ pontban felvett Φ függvényérték (vagyis $x_i < 0$ legyen), a pontok másik kétharmad részére pedig $x_i \geq 0$ teljesüljön,

(ii) az intervallum legyen elég széles ahhoz, hogy a kapott regressziós becslés stabil legyen, például $f(a) - f(b) \geq \sigma$ teljesüljön,

(iii) az $x_j \in [a,b]$ mintapontokat egyenletesen vesszük fel az intervallumon.

A példák számozása a [De 98a] cikkben leírt példák esetén megadott számozást követi; néhány példán a komponensenkénti eljárás numerikus viselkedését mutatjuk be. Egy $f(x_i) \sim p_i$ függvénymeghatározás szórása $\sigma = 0.05$ volt, a mintaszám $n = 25$ volt.

A 4. példa jellemzői a következők voltak: $m = 50$ dimenziós eloszlás, $\mathbf{h}$ adott, $\Phi(\mathbf{h}) = 0.95$, $f_1(x) = \Phi(\mathbf{h} + x\mathbf{e}_1)$, a $\nabla\Phi(\mathbf{h})$ gradiens első komponensének a „pontos” értéke 0.0883. A „pontos” értéket itt is és a továbbiakban is úgy határoztuk meg, hogy a mintavételt ismételten elvégeztük $\sigma/10$ hibával $n = 1000$ mintapont esetén. A kapott numerikus

lineáris	alap	logaritmikus	ford.-logar.
(0.086,0.120)	(0.086,0.120)	(0.085,0.120)	(0.091,0.118)

4.6. táblázat. Gradiens becslés, 1. példa, különböző becslések gradiensei.

lineáris	alap	logaritmikus	ford.-logar.
0.139	0.139	0.140	0.137
0.164	0.164	0.171	0.163

4.7. táblázat. Gradiens becslés, 3. példa, 2 különböző futás.

eredményeket a 4.4 táblázatban adjuk meg, amelyek azt szemléltetik, hogyan ingadozik a gradiens első komponensének becslése az $[a, b]$ intervallum változtatásával. A táblázatban a „lineáris" elnevezésű oszlop azokat az eredményeket tartalmazza, amelyeket egy $t_0(x) = a_0x + b_0$ alakú regresszióval kaptunk.

Az 1. példa egy kétdimenziós normális eloszlásfüggvény, $\mathbf{h} = (1.1114, 1.5), \Phi(\mathbf{h}) = 0.8$ értékekkel, a korrelációs együttható $\varrho = -0.9$, a gradiens első komponensének „pontos" értéke 0.215. A 4.5 táblázatban az erre a példára, ugyanazon paraméterek esetén, de különböző futások során (másmilyen véletlen minták felhasználásával) kapott eredményeket adjuk meg.

Ugyanezen példára vonatkozó eredményeket közlünk a 4.6 táblázatban, ahol $\mathbf{h} = (1.8358, 1.5), \Phi(\mathbf{h}) = 0.9$, a gradiens „pontos" értéke (0.074,0.129). A 3. példa esetére kapott számítógépes eredményeket adjuk meg az utolsó táblázatban (itt $m = 10$ dimenziós eloszlásunk volt, $\mathbf{h}$ adott, $\Phi(\mathbf{h}) = 0.8$, $f_1(x) = \Phi(\mathbf{h} + x\mathbf{e}_1)$, a gradiens első komponensének „pontos" értéke 0.1405).

A gyökkeresésre kidolgozott számítógépes program részletei a [De 98d] cikkben találhatók meg, itt további számítógépes futások eredményei is megtalálhatók.

A fenti eredményekből és más, itt nem közölt számítógépes eredményekből a következő sejtést lehet megkockáztatni. A gradiens kiszámításának a hibája $3\sigma/\sqrt{n}$ nagyságúnak tűnik, tehát a Prékopa által javasolt módszer jobbnak tűnik (annak ellenére, hogy új korrelációs mátrixokat kell számolni a feltételes eloszlásokhoz). Előnyös lehet viszont az egydimenziós becslések használata, illetőleg iránymenti deriváltak számítása, ha például egy jó leszálló irányra van szükségünk egy véletlen kereső algoritmusban, hiszen ilyenkor nem kell a teljes gradienst kiszámítanunk. A fentebbi regressziós becslések segítségével történő gradiens számítás mindenképpen stabilabb a numerikus differencia használatánál, hiszen esetenként az könnyen előjelet is válthat.

5. fejezet

Szukcesszív regressziós approximációk egydimenzióban

Sok numerikus feladat visszavezethető egy

$$f(x) = 0 \tag{5.1}$$

egyenlet gyökének numerikus meghatározására, ahol $f : R^1 \to R^1$. Ennek a gyakran fellépő feladatnak sok megoldó algoritmusa ismeretes, lásd például [Lue 84], vagy [BSS 94], [AF 01]). Ha az $f(x)$ értékét pontosan meg tudjuk határozni (determinisztikus eset), akkor általában jól alkalmazhatók ezek az eljárások. Ha a függvényértékek kiszámítása csak egy véletlen additív hibával lehetséges (zajos függvényérték), akkor nehezebb a helyzet. A fejezet első részében a determinisztikus esettel, a második részben a zajos függvényértékek esetével foglalkozunk.

Először leírjuk, hogyan lehet a legkisebb négyzetek módszerével meghatározni egy közelítést, majd egy szukcesszív regressziós approximációnak (SRA) nevezett, iteratív eljárást írjuk le a gyök meghatározására. Bebizonyítjuk az eljárás konvergenciáját determinisztikus függvénykiszámítás esetére a [De 01a] cikk alapján: az SRA egy olyan pontsorozatot állít elő, amely az (5.1) egyenlet gyökéhez konvergál. Ez az eljárás nem konvergál olyan gyorsan, mint az ismert eljárások; igazi haszna a zajos függvények, illetőleg az eljárásnak a következő fejezetben leírt többdimenziós általánosításában rejlik.

A fejezet második szakaszában a zajos függvényértékek kiszámítása esetére megadjuk a szukcesszív regressziós algoritmus sztochasztikus változatát és az algoritmus néhány tulajdonságát bizonyítjuk. Végül néhány, a [De 01b] cikkben közölt számítási eredménnyel szemléltetjük az algoritmus működését.

5.1. Determinisztikus függvényérték

5.1.1. Jelölések és az SRA_D algoritmus

Tegyük fel egyelőre, hogy az $f(x)$ függvény értékeit tetszőleges x esetén pontosan ki tudjuk számítani – ezt az esetet a determinisztikus függvényérték esetének nevezzük, valamint azt, hogy a következő feltevés igaz:

$A(f-1)$ $f(x), x \in R^1$ egy folytonos függvény, amely tetszőleges két x_i, x_j pont esetén a $0 < \delta_L \leq \frac{f(x_j)-f(x_i)}{x_j-x_i} \leq \delta_U < \infty$ egyenlőtlenségek fennállnak valamilyen δ_L, δ_U állandókkal.

A feltevésből következik, hogy az $f(x) = 0$ egyenletnek egyetlen Θ gyöke van, azaz

$$f(\Theta) = 0,$$

továbbá $f(x) < 0$, ha $x < \Theta$ (és $f(x) > 0$, ha $x > \Theta$). Miután a javasolt SRA algoritmus által előállított összes $\{x_i\}$ pont egy korlátos intervallumon belül helyezkedik el, elég az $A(f-1)$ feltevést egy elég nagy intervallumon belül megkövetelni (ez a korlátosság bebizonyítható, vagy feltehető).

Közelítsük az $f(x)$ függvényt egy lineáris függvénnyel, amelyet a legkisebb négyzetek módszerével határozunk meg. Adott k darab x_i pont és $f_i = f(x_i)$ függvényérték esetén keressük a minimális L_2 normájú, $g_k(x) = \alpha_k x + \beta_k$ alakú közelítést, vagyis adott $S_k = \{x_i, f_i\}_{i=0}^{k-1}$ esetén legyen $g_k(x)$ a következő feladat megoldása:

$$\min_{\alpha_k,\beta_k} \sum_{i=0}^{k-1} \left[f_i - (\alpha_k x_i + \beta_k)\right]^2.$$

Az optimalitás elsőrendű szükséges feltételei által adott egyenletrendszert az előző összeg α_k, β_k szerinti deriválásával kaphatjuk meg:

$$\begin{aligned} \sum_{i=0}^{k-1} x_i \left[f_i - (\alpha_k x_i + \beta_k)\right] &= 0, \\ \sum_{i=0}^{k-1} \left[f_i - (\alpha_k x_i + \beta_k)\right] &= 0, \end{aligned} \tag{5.2}$$

amit az ismeretlen α_k, β_k paraméterekre kell megoldani. Vezessük be a következő jelöléseket:

$$\begin{aligned} m_0 = \tfrac{1}{k}\textstyle\sum_{i=0}^{k-1} f_i, \quad m_1 = \tfrac{1}{k}\textstyle\sum_{i=0}^{k-1} x_i f_i, \\ M_0 = \tfrac{1}{k}\textstyle\sum_{i=0}^{k-1} 1 = 1, \quad M_1 = \tfrac{1}{k}\textstyle\sum_{i=0}^{k-1} x_i, \; M_2 = \tfrac{1}{k}\textstyle\sum_{i=0}^{k-1} x_i^2, \end{aligned}$$

ezek felhasználásával (5.2) átírható az

$$\begin{aligned} m_1 &= \alpha_k M_2 + \beta_k M_1, \\ m_0 &= \alpha_k M_1 + \beta_k M_0 \end{aligned}$$

alakba, amiből a $g_k(x)$ függvény keresett paraméterei meghatározhatóak:

$$\alpha_k = \frac{-m_0 M_1 + m_1}{M_2 - M_1^2}, \tag{5.3}$$

$$\beta_k = \frac{m_0 M_2 - m_1 M_1}{M_2 - M_1^2}. \tag{5.4}$$

A közelítő $g_k(x) = \alpha_k x + \beta_k$ függvény $g_k(x) = 0$ egyenlőséget kielégítő gyöke pedig

$$x_k = -\frac{\beta_k}{\alpha_k} = -\frac{m_0 M_2 - m_1 M_1}{-m_0 M_1 + m_1}. \tag{5.5}$$

Természetesen nemcsak α_k, β_k, g_k függ k-tól, de egyszerűség kedvéért eltekintünk a többi mennyiség indexelésétől (hacsak az kifejezetten nem szükséges).

Megadjuk a szukcesszív regressziós approximációk módszerének az Θ gyök meghatározására szolgáló eljárását. A lényeges pontja ennek az eljárásnak, hogy az újonnan kiszámított közelítő gyököt (valamint a függvényértéket) hozzáadjuk az S_k eddigi ponthalmazhoz, és az új közelítést ennek a kibővített halmaznak a segítségével határozzuk meg.

Az algoritmus formális leírása a következő:

SRA_D - egydimenziós gyökkeresés, determinisztikus függvény

0. Tegyük fel, hogy rendelkezésünkre áll egy kiindulási $S_k = \{x_i, f_i\}_{i=0}^{k-1}$ halmaz és legyen a k iterációs számláló az adott pontok száma.
1. Számítsuk ki a $g_k(x) = \alpha_k x + \beta_k$ együtthatóit az S_k-ból.
2. Határozzuk meg az x_k közelítő gyököt az $g_k(x) = 0$ egyenletből.
3. Ha x_k „elég jó'', akkor STOP. Egyébként számítsuk ki az $f_k = f(x_k)$ függvényértéket és legyen $S_{k+1} = S_k \cup \{x_k, f_k\}$, továbbá $k = k + 1$, és menjünk vissza az 1. lépésre.

Az algoritmusban szereplő „elég jó" kifejezés értelmezését későbbi változatokban fogjuk megadni (lásd a 6.2.4 pontot és a 6.7 szakaszt). A közelítő gyöknek a ponthalmazhoz való hozzáadása egyfajta visszacsatolásnak is értelmezhető, így az algoritmus visszacsatolt regressziónak is nevezhető. Megjegyezzük, hogy az S_k halmazban ugyanaz a pont többször is előfordulhat, a többszörös előfordulás viszont befolyásolja a g_k közelítés paramétereit. Az alábbi levezetésekhez (illetőleg a paraméterek fenti módon való megoldhatóságához) csak azt kell feltennünk, hogy legalább két különböző pont van S_k-ban.

5.1.2. Néhány tulajdonság

Az alábbiakban a gyökkeresésben szereplő mennyiségekre vonatkozó néhány hasznos tulajdonságot állapítunk meg.

7. Lemma. *Tegyük fel, hogy S_k adott, akkor $m_0 > 0$ esetén $x_k < M_1$ (és $m_0 < 0$-ból következik, hogy $x_k > M_1$).*

Bizonyítás. A következő egyszerű képletet vezetjük le először:

$$\begin{aligned} x_k &= -\frac{\beta_k}{\alpha_k} = -\frac{m_0M_2 - m_1M_1}{-m_0M_1 + m_1} = -\frac{m_0M_2 - m_0M_1^2 + m_0M_1^2 - m_1M_1}{-m_0M_1 + m_1} \\ &= -\frac{m_0(M_2 - M_1^2) - M_1(-m_0M_1 + m_1)}{-m_0M_1 + m_1} = \frac{-m_0}{\frac{-m_0M_1+m_1}{M_2-M_1^2}} + M_1 = M_1 - \frac{m_0}{\alpha_k}. \end{aligned} \tag{5.6}$$

Ez a kifejezés bizonyítja a lemmában foglalt állítást, ha $\alpha_k > 0$; ennek az utóbbi egyenlőtlenségnek a fennállását a következő tétel (ii) és (iii) részei mutatjuk meg. □.

Jegyezzük meg, hogy a közelítő $y = \alpha_k x + \beta_k$ egyenes átmegy $(x_k, 0)$ és a (M_1, m_0) pontokon – ez például közvetlen behelyettesítéssel is ellenőrizhető. Átírjuk a gyökkeresés folyamán felhasznált különböző mennyiségeket, hogy időnként jobban használható kifejezéseket kapjunk.

8. Tétel. *Tekintsünk egy f függvényt, amelyre fennáll az $A(f-1)$ feltevés és egy adott $S_k = \{x_i, f_i\}_{i=0}^{k-1}$ halmazt. Jelölje az (x_i, f_i) és az (x_j, f_j) pontokat összekötő $l_{ij}(x) = \alpha_{ij}x + \beta_{ij}$ egyenesnek a Θ_{ij} gyökét. Ekkor*

(i)

$$m_0M_2 - m_1M_1 = \frac{1}{k^2}\sum_{i=0}^{k-2}\sum_{j=i+1}^{k-1}(x_j - x_i)(x_jf_i - x_if_j),$$

(ii)

$$-m_0M_1 + m_1 = \frac{1}{k^2}\sum_{i=0}^{k-2}\sum_{j=i+1}^{k-1}(x_j - x_i)(f_j - f_i) > 0,$$

(iii)

$$M_2 - M_1^2 = \frac{1}{k^2}\sum_{i=0}^{k-2}\sum_{j=i+1}^{k-1}(x_j - x_i)^2 > 0,$$

(iv)

$$x_k = -\frac{\sum_{i=0}^{k-2}\sum_{j=i+1}^{k-1}(x_j - x_i)(x_jf_i - x_if_j)}{\sum_{i=0}^{k-2}\sum_{j=i+1}^{k-1}(x_j - x_i)(f_j - f_i)},$$

(v)

$$x_k = \sum_{i=0}^{k-2}\sum_{j=i+1}^{k-1}\Theta_{ij}w_{ij}^{(k)}, \quad \alpha_k = \sum_{i=0}^{k-2}\sum_{j=i+1}^{k-1}\alpha_{ij}\lambda_{ij}^{(k)}, \quad \beta_k = \sum_{i=0}^{k-2}\sum_{j=i+1}^{k-1}\beta_{ij}\lambda_{ij}^{(k)},$$

ahol $w_{ij}^{(k)} \geq 0,\ \sum_{i=0}^{k-2}\sum_{j=i+1}^{k-1} w_{ij}^{(k)} = 1,\ \lambda_{ij}^{(k)} \geq 0,\ \sum_{i=0}^{k-2}\sum_{j=i+1}^{k-1}\lambda_{ij}^{(k)} = 1.$

Megjegyzés. A továbbiakban fogjuk használni a fentebbi (iv), tört-formában megadott x_k számlálójának és nevezőjének, valamint a (iii) alatti kifejezésnek egy többszörösét, ezért bevezetjük a következő jelöléseket is:

$$\begin{aligned} num(k) &= k^2[m_0M_2 - m_1M_1] = \textstyle\sum_{i=0}^{k-2}\sum_{j=i+1}^{k-1}(x_j - x_i)(x_jf_i - x_if_j), \\ den(k) &= k^2[-m_0M_1 + m_1] = \textstyle\sum_{i=0}^{k-2}\sum_{j=i+1}^{k-1}(x_j - x_i)(f_j - f_i) > 0, \\ det(k) &= k^2[M_2 - M_1^2] = \textstyle\sum_{i=0}^{k-2}\sum_{j=i+1}^{k-1}(x_j - x_i)^2. \end{aligned}$$

Bizonyítás. A tétel állításainak belátásához csak az összegek átírására van szükség. (i) Az m_0, m_1, M_1, M_2 kifejezések definícióját felhasználva kapjuk az x_k törtjének számlálóját:

$$\begin{aligned} k^2[m_0M_2 - m_1M_1] &= k^2\left[\left(\tfrac{1}{k}\textstyle\sum_{i=0}^{k-1} f_i\right)\left(\tfrac{1}{k}\sum_{j=0}^{k-1} x_j^2\right) - \left(\tfrac{1}{k}\sum_{i=0}^{k-1} x_if_i\right)\left(\tfrac{1}{k}\sum_{j=0}^{k-1} x_j\right)\right] \\ &= \textstyle\sum_{i=0}^{k-1}\sum_{j=0}^{k-1} x_j^2f_i - x_ix_jf_i \\ &= \textstyle\sum_{i=0}^{k-2}\sum_{j=i+1}^{k-1} x_j^2f_i - x_ix_jf_i + x_i^2f_j - x_ix_jf_j \\ &= \textstyle\sum_{i=0}^{k-2}\sum_{j=i+1}^{k-1}(x_j - x_i)(x_jf_i - x_if_j), \end{aligned}$$

ahol a második sorban felhasználtuk, hogy $x_j^2f_i - x_ix_jf_i = 0$ az $i = j$ esetén.

(ii) Az előzőhöz hasonlóan kapható az x_k nevezőjének kifejezése:

$$\begin{aligned} k^2[-m_0M_1 + m_1] &= k^2\left[-\left(\tfrac{1}{k}\textstyle\sum_{i=0}^{k-1} f_i\right)\left(\tfrac{1}{k}\sum_{j=0}^{k-1} x_j\right) + \left(\tfrac{1}{k}\sum_{i=0}^{k-1} x_if_i\right)\right] \\ &= \textstyle\sum_{i=0}^{k-1}\sum_{j=0}^{k-1} x_if_j - x_jf_i \\ &= \textstyle\sum_{i=0}^{k-2}\sum_{j=i+1}^{k-1} x_if_i - x_jf_i + x_jf_j - x_if_j \\ &= \textstyle\sum_{i=0}^{k-2}\sum_{j=i+1}^{k-1}(x_j - x_i)(f_j - f_i), \end{aligned}$$

mivel az azonos $i = j$ indexű tagok összevonva eltünnek. Itt $x_j > x_i$ esetén $f_j > f_i$ a függvény monoton növekvése miatt. Tehát az utolsó kettős összegben minden tag pozitív, tehát a $-m_0M_1 + m_1 = \frac{1}{k^2}den(k)$ is pozitív.

(iii) Az $M_2 - M_1^2$ tag nemnegatívitása és adott formája a következőkből látható:

$$\begin{aligned} k^2[M_2 - M_1^2] &= k^2\left[\left(\tfrac{1}{k}\textstyle\sum_{i=0}^{k-1} x_i^2\right) - \left(\tfrac{1}{k}\sum_{i=0}^{k-1} x_i\right)\left(\tfrac{1}{k}\sum_{j=0}^{k-1} x_j\right)\right] \\ &= \textstyle\sum_{i=0}^{k-1}\sum_{j=0}^{k-1} x_i^2 - x_ix_j \\ &= \textstyle\sum_{i=0}^{k-2}\sum_{j=i+1}^{k-1} x_i^2 - x_ix_j + x_j^2 - x_ix_j \\ &= \textstyle\sum_{i=0}^{k-2}\sum_{j=i+1}^{k-1}(x_j - x_i)^2. \end{aligned}$$

(iv) Az előbbi (i) és (ii) kifejezésből kaphatjuk az x_k-ra vonatkozó egyenlőséget:

$$x_k = -\frac{m_0M_2 - m_1M_1}{-m_0M_1 + m_1} = -\frac{\sum_{i=0}^{k-2}\sum_{j=i+1}^{k-1}(x_j - x_i)(x_jf_i - x_if_j)}{\sum_{i=0}^{k-2}\sum_{j=i+1}^{k-1}(x_j - x_i)(f_j - f_i)} = -\frac{num(k)}{den(k)}.$$

(v) Tekintsük az $l_{ij}(x) = \alpha_{ij}x + \beta_{ij}$ egyenest, melynek egyenlete

$$l_{ij}(x) = \frac{f_j - f_i}{x_j - x_i}x + \frac{x_j f_i - x_i f_j}{x_j - x_i}.$$

Az $l_{ij}(x) = 0$ egyenlet Θ_{ij} gyöke és az egyenes paraméterei könnyen meghatározhatók:

$$\Theta_{ij} = -\frac{x_j f_i - x_i f_j}{f_j - f_i},\ \alpha_{ij} = \frac{f_j - f_i}{x_j - x_i},\ \beta_{ij} = \frac{x_j f_i - x_i f_j}{x_j - x_i}.$$

A (iv) részben levezetett és a Θ_{ij} kifejezést használva az x_k keresett kifejezése megkapható:

$$\begin{aligned} x_k &= -\frac{m_0 M_2 - m_1 M_1}{-m_0 M_1 + m_1} = -\frac{\sum_{i=0}^{k-2}\sum_{j=i+1}^{k-1}(x_j - x_i)(x_j f_i - x_i f_j)}{\sum_{i=0}^{k-2}\sum_{j=i+1}^{k-1}(x_j - x_i)(f_j - f_i)} \\ &= \sum_{i=0}^{k-2}\sum_{j=i+1}^{k-1}\left[-\frac{x_j f_i - x_i f_j}{f_j - f_i}\ \frac{(x_j - x_i)(f_j - f_i)}{\sum_{i=0}^{k-2}\sum_{j=i+1}^{k-1}(x_j - x_i)(f_j - f_i)}\right] \\ &= \sum_{i=0}^{k-2}\sum_{j=i+1}^{k-1}\Theta_{ij} w_{ij}^{(k)}, \end{aligned} \tag{5.7}$$

ahol minden $w_{ij}^{(k)} = \frac{(x_j - x_i)(f_j - f_i)}{\sum_{i=0}^{k-2}\sum_{j=i+1}^{k-1}(x_j - x_i)(f_j - f_i)}, i = 0, 1, \ldots, k-2, j = i+1, \ldots, k-1$ súly nemnegatív és természetesen

$$\sum_{i=0}^{k-2}\sum_{j=i+1}^{k-1} w_{ij}^{(k)} = 1.$$

Tehát a gyökkereső SRA_D algoritmus által szolgáltatott x_k közelítő gyök konvex kombinációja a minden lehetséges módon vett (x_i, f_i) és (x_j, f_j) pontpárokat összekötő $l_{ij}, i = 0, 1, \ldots, k-2,\ j = i+1, \ldots, k-1$ egyenesek gyökeinek. A normáló faktortól eltekintve a súly értéke $(x_j - x_i)(f_j - f_i)$, tehát minél messzebb vannak a Θ_{ij} gyököt megadó pontpár tagjai egymástól, annál nagyobb súllyal szerepel az általuk megadott gyök.

A hátralévő képleteket hasonlóan lehet levezetni, ezeknél a $\lambda_{ij} = \frac{(x_j-x_i)^2}{\sum_{i=0}^{k-2}\sum_{j=i+1}^{k-1}(x_j-x_i)^2}$, $\lambda_{ij} > 0$ jelöléseket használtuk:

$$\begin{aligned}
\alpha_k = & \sum_{i=0}^{k-2}\sum_{j=i+1}^{k-1}\frac{(x_j-x_i)(f_j-f_i)}{\sum_{i=0}^{k-2}\sum_{j=i+1}^{k-1}(x_j-x_i)^2} \\
= & \sum_{i=0}^{k-2}\sum_{j=i+1}^{k-1}\frac{((f_j-f_i)}{(x_j-x_i)}\frac{(x_j-x_i)^2}{\sum_{i=0}^{k-2}\sum_{j=i+1}^{k-1}(x_j-x_i)^2} = \sum_{i=0}^{k-2}\sum_{j=i+1}^{k-1}\alpha_{ij}\lambda_{ij}, \\
\beta_k = & \frac{1}{k^2}\sum_{i=0}^{k-2}\sum_{j=i+1}^{k-1}\frac{(x_j-x_i)(x_jf_i-x_if_j)}{\sum_{i=0}^{k-2}\sum_{j=i+1}^{k-1}(x_j-x_i)^2} \\
= & \sum_{i=0}^{k-2}\sum_{j=i+1}^{k-1}\frac{(x_jf_i-x_if_j)}{(x_j-x_i)}\frac{(x_j-x_i)^2}{\sum_{i=0}^{k-2}\sum_{j=i+1}^{k-1}(x_j-x_i)^2} = \sum_{i=0}^{k-2}\sum_{j=i+1}^{k-1}\beta_{ij}\lambda_{ij}.
\end{aligned}$$

Tehát nemcsak a Θ gyök, hanem az α_k, β_k paraméterek is konvex kombinációkként állíthatók elő; az α_{ij} (illetőleg a β_{ij}) súlya csak az $(x_j - x_i)^2$ távolságnégyzettel arányos, a függvényértékektől nem függ. Ebből is triviálisan adódik a 7. lemmában felhasznált $\alpha_k > 0$ egyenlőtlenség, hiszen még $\alpha_{ij} \geq \delta_L > 0$ is igaz.

A továbbiakban az SRA_D által, egy $S_{k_0} = \{x_i, f_i\}_{i=0}^{k_0-1}$, k_0 számú pontot és függvényértéket tartalmazó kezdeti halmazból kiindulva számított $\{x_i\}_{i=k_0}^{\infty}$ pontsorozatának két tetszőleges, egymásutáni x_k és x_{k+1} pontját egyszerűen egymásutáni pontoknak nevezzük csak – nyilvánvalóan x_k-t $S_k = \{x_i, f_i\}_{i=0}^{k-1}$-ból, x_{k+1}-et $S_{k+1} = S_k \cup \{x_k, f_k\}$-ból számítjuk az SRA_D algoritmussal, és $k > k_0 \geq 2$. Az SRA_D által két, egymás után előállított x_k és x_{k+1} pont közötti összefüggést írjuk le:

9. Tétel. *Legyen $S_k = \{x_i, f_i\}_{i=0}^{k-1}$ adott, x_k és x_{k+1} egymásutáni pontok, akkor*

$$x_{k+1} = x_k - f_k\sum_{i=0}^{k-1}\frac{(x_k-x_i)^2}{den(k+1)} = x_k - f_k\frac{\sum_{i=0}^{k-1}(x_k-x_i)^2}{\sum_{i=0}^{k-1}\sum_{j=i+1}^{k}(x_j-x_i)(f_j-f_i)}.$$

Bizonyítás. Tekintsük az x_k-ra az előző tétel (v) részében adott kifejezést és helyettesítsük k-t $(k+1)$-gyel:

$$\begin{aligned}
x_{k+1} = & \sum_{i=0}^{k-1}\sum_{j=i+1}^{k}\left[-\frac{x_jf_i-x_if_j}{f_j-f_i}\ \frac{(x_j-x_i)(f_j-f_i)}{den(k+1)}\right] = \\
= & \sum_{i=0}^{k-2}\sum_{j=i+1}^{k-1}\left[-\frac{x_jf_i-x_if_j}{f_j-f_i}\ \frac{(x_j-x_i)(f_j-f_i)}{den(k+1)}\right] + \sum_{i=0}^{k-1}\left[-\frac{x_kf_i-x_if_k}{f_k-f_i}\ \frac{(x_k-x_i)(f_k-f_i)}{den(k+1)}\right] \\
= & \frac{den(k)}{den(k+1)}x_k\ +\ \sum_{i=0}^{k-1}\left[-\frac{x_kf_i-x_if_k}{f_k-f_i}\right]\ \frac{(x_k-x_i)(f_k-f_i)}{den(k+1)} \qquad (5.8) \\
= & \frac{den(k)}{den(k+1)}x_k + \sum_{i=0}^{k-1}\Theta_{ik}w_{ik}^{k+1}.
\end{aligned}$$

Egy adott $i, i = 0, 1, \ldots, k-1$ esetén a Θ_{ik} gyök kifejezése átírható:

$$\Theta_{ik} = -\frac{x_k f_i - x_i f_k}{f_k - f_i} = x_k - f_k \frac{x_k - x_i}{f_k - f_i}.$$

Ezt a (5.8) egyenletbe helyettesítve a tétel állítását kapjuk, hiszen

$$\begin{aligned} x_{k+1} &= \frac{den(k)}{den(k+1)} x_k + x_k \sum_{i=0}^{k-1} \frac{(x_k - x_i)(f_k - f_i)}{den(k+1)} - f_k \sum_{i=0}^{k-1} \frac{x_k - x_i}{f_k - f_i} \frac{(x_k - x_i)(f_k - f_i)}{den(k+1)} \\ &= x_k - f_k \sum_{i=0}^{k-1} \frac{(x_k - x_i)^2}{den(k+1)}. \square \end{aligned}$$

A tétel állításából adódó, $x_{k+1} - x_k = -f_k \frac{\sum_{i=0}^{k-1}(x_k - x_i)^2}{den(k+1)}$ mennyiséget az algoritmus lépéshosszának nevezzük – ezt a mennyiséget a pontsorozat korlátosságának bizonyításában fogjuk használni. Ennek az utolsó tételnek egy egyszerű, de fontos következménye az alábbi:

10. Tétel. *Ha $x_k < \Theta$, akkor $x_k < x_{k+1}$ (illetőleg $x_k > \Theta$ esetén $x_k > x_{k+1}$).*

Bizonyítás. Az előző tétel eredménye alapján írhatjuk, hogy

$$x_{k+1} - x_k = -f_k \sum_{i=0}^{k-1} \frac{(x_k - x_i)^2}{den(k+1)}.$$

Ha $x_k < \Theta$, akkor $f_k < 0$, továbbá a 8. tétel (ii) része alapján $den(k+1) > 0$, továbbá $(x_k - x_i)^2 > 0$ triviálisan, tehát az összeg minden tagja pozitív, következésképpen $x_{k+1} - x_k > 0$, ha $f_k < 0$ (illetőleg $x_k > \Theta$ esetén $f_k > 0$, tehát ugyanúgy $x_{k+1} < x_k$).$\square$

Következmény. Ha $f_k = 0$, akkor az előző tétel alapján $x_{k+1} - x_k = 0, x_{n+1} - x_n = 0$ vagyis $f_n = f(x_n) = 0$, tehát $x_n = \Theta$ minden $n = k, k+1, \ldots$ indexre, vagyis ha egyszer az algoritmus megtalálta a valódi gyököt, akkor nem változik meg – a Θ az SRA_D algoritmus fixpontja.

11. Tétel. *Legyen $S_k = \{x_i, f_i\}_{i=0}^{k-1}$ adott, x_k és x_{k+1} egymásutáni pontok. Jelölje az S_{k+1} halmazból meghatározott közelítő függvényt $g_{k+1}(x)$, akkor*

$$x_{k+1} = x_k - \frac{g_{k+1}(x_k)}{g'_{k+1}(x_k)}. \tag{5.9}$$

Bizonyítás. Vegyük észre, hogy

$$f_k \sum_{i=0}^{k-1} (x_k - x_i)^2 = [num(k+1) - num(k)] + [den(k+1) - den(k)] x_k,$$

akkor az $x_{k+1}-x_k$ lépéshossz 10. tételben szereplő kifejezést átírhatjuk a következőképpen: osszuk el a $det(k+1)$ kifejezéssel a tört számlálóját és nevezőjét is, valamint vegyük figyelembe, hogy $g_k(x_k) = \alpha_k x_k + \beta_k = \frac{den(k)}{det(k)} x_k + \frac{num(k)}{det(k)} = 0$, ekkor

$$\begin{aligned} x_{k+1} - x_k &= -\frac{[num(k+1) - num(k)] + [den(k+1) - den(k)]x_k}{den(k+1)} \\ &= -\frac{\frac{num(k+1)}{det(k+1)} - \frac{num(k)}{det(k+1)} + \frac{den(k+1)}{det(k+1)} x_k - \frac{den(k)}{det(k+1)} x_k}{\frac{den(k+1)}{det(k+1)}} \\ &= -\frac{\beta_{k+1} - \frac{det(k)}{det(k+1)}\beta_k + \alpha_{k+1} x_k - \frac{det(k)}{det(k+1)}\alpha_k x_k}{\alpha_{k+1}} \\ &= -\frac{g_{k+1}(x_k) - \frac{det(k)}{det(k+1)} g_k(x_k)}{\alpha_{k+1}} = -\frac{g_{k+1}(x_k)}{g'_{k+1}(x_k)}. \square \end{aligned}$$

Ez a zárt forma (ami egyébként minden lineáris közelítésre igaz) a Newton-Raphson formulára emlékeztet, de vegyük észre, hogy itt a kifejezésben szereplő g_{k+1} függvény iterációról iterációra változik, továbbá az előzőleg kiszámított ponttól (pontoktól) függ.

5.1.3. A közelítés paramétereinek újraszámítása

Legyen adott S_k, amelyből a közelítő $g_k(x) = \alpha_k x + \beta_k$ függvény paramétereit a

$$\begin{aligned} m_0 &= \alpha_k M_1 + \beta_k M_0, \\ m_1 &= \alpha_k M_2 + \beta_k M_1, \end{aligned} \tag{5.10}$$

egyenletrendszer megoldásával lehet meghatározni. Azzal a kérdéssel foglalkozunk most, hogy a következő iterációban kiszámítandó $\alpha_{k+1}, \beta_{k+1}$ paramétereket hogyan lehet az előző α_k, β_k segítségével hatékonyan meghatározni. A feladat lényegében egy mátrix inverzének felfrissítésére használt eljárásra vezethető vissza.

Az $S_{k+1} = S_k \cup \{x_k, f_k\}$ halmazból kiszámított mennyiségeket ebben a szakaszban $(k+1)$ felső indexszel fogjuk megkülönböztetni az S_k halmazból számított mennyiségektől (amelyeknek nem adunk külön felső indexet). Jelölje a (5.10) egyenlőség jobboldalának a k-adik iterációhoz tartozó együtthatómátrixszát $\mathbf{M}$, ennek segítségével a megoldandó egyenletrendszer és a megoldás a következő formába írható:

$$\begin{aligned} \mathbf{M} &= \begin{pmatrix} M_0 & M_1 \\ M_1 & M_2 \end{pmatrix}, \begin{pmatrix} m_0 \\ m_1 \end{pmatrix} = \mathbf{M} \begin{pmatrix} \beta_k \\ \alpha_k \end{pmatrix}, \\ \mathbf{M}^{-1} &= \frac{1}{|\mathbf{M}|} \begin{pmatrix} M_2 & -M_1 \\ -M_1 & M_0 \end{pmatrix}, \begin{pmatrix} \beta_k \\ \alpha_k \end{pmatrix} = \mathbf{M}^{-1} \begin{pmatrix} m_0 \\ m_1 \end{pmatrix}, \end{aligned}$$

ahol $|\mathbf{M}| = M_2 - M_1^2$ a determináns értéke. Legyen most $\mathbf{x}' = (1, x_k)$, akkor az m_0, m_1, M_0, M_1, M_2 mennyiségek, valamint az $\mathbf{M}^{(k+1)}$ mátrix új értéke:

$$
\begin{aligned}
m_0^{(k+1)} &= \frac{1}{k+1}\sum_{i=0}^{k} f_i = \frac{k}{k+1}m_0 + \frac{1}{k+1}f_k,\\
m_1^{(k+1)} &= \frac{k}{k+1}m_1 + \frac{1}{k+1}x_k f_k,\\
M_0^{(k+1)} &= \frac{k}{k+1}M_0 + \frac{1}{k+1}1,\\
M_1^{(k+1)} &= \frac{k}{k+1}M_1 + \frac{1}{k+1}x_k,\\
M_2^{(k+1)} &= \frac{k}{k+1}M_2 + \frac{1}{k+1}x_k^2,\\
\mathbf{M}^{(k+1)} &= \frac{k}{k+1}\mathbf{M} + \frac{1}{k+1}\begin{pmatrix} 1 & x_k \\ x_k & x_k^2 \end{pmatrix} = \frac{k}{k+1}\mathbf{M} + \frac{1}{k+1}\mathbf{x}\mathbf{x}'
\end{aligned}
$$

Mivel az új $\mathbf{M}^{(k+1)}$ értékét megadó összeg második tagja egy diadikus szorzat, ezért az

$$(A + \mathbf{u}\mathbf{v}')^{-1} = A^{-1} - \frac{A^{-1}\mathbf{u}\mathbf{v}'A^{-1}}{1 + \mathbf{v}'A^{-1}\mathbf{u}}$$

Sherman-Morrison formulát lehet használni az $\mathbf{M}^{(k+1)}$ inverzének meghatározására:

$$\left[\mathbf{M}^{(k+1)}\right]^{-1} = \left[\frac{k}{k+1}\mathbf{M} + \frac{1}{k+1}\mathbf{x}\mathbf{x}'\right]^{-1} = \frac{k+1}{k}\left\{\mathbf{M}^{-1} - \frac{1}{k}\frac{\mathbf{M}^{-1}\mathbf{x}\mathbf{x}'\mathbf{M}^{-1}}{1 + \frac{1}{k}\mathbf{x}'\mathbf{M}^{-1}\mathbf{x}}\right\}.$$

Ezek szerint az új $\mathbf{M}^{(k+1)}$mátrix inverze a régi mátrix inverzének és egy diadikus szorzatnak az összege. Ennek segítségével az új közelítés $\alpha_{k+1}, \beta_{k+1}$ paraméterei a következő formában adhatók meg:

$$
\begin{aligned}
\begin{pmatrix} \beta_{k+1} \\ \alpha_{k+1} \end{pmatrix} &= \left[\mathbf{M}^{(k+1)}\right]^{-1}\begin{pmatrix} m_0^{(k+1)} \\ m_1^{(k+1)} \end{pmatrix}\\
&= \left[\mathbf{M}^{(k+1)}\right]^{-1}\left\{\frac{k}{k+1}\begin{pmatrix} m_0 \\ m_1 \end{pmatrix} + \frac{1}{k+1}\begin{pmatrix} f_k \\ x_k f_k \end{pmatrix}\right\}.
\end{aligned}
$$

Bevezetve a $\mathbf{D}$ jelölést a diadikus szorzatra, vagyis legyen

$$\mathbf{D} = \frac{\mathbf{M}^{-1}\mathbf{x}\mathbf{x}'\mathbf{M}^{-1}}{1 + \frac{1}{k}\mathbf{x}'\mathbf{M}^{-1}\mathbf{x}},$$

az új paraméterértékeket a régiekből a következőképpen lehet újraszámítani:

$$\begin{aligned}\begin{pmatrix}\beta_{k+1}\\ \alpha_{k+1}\end{pmatrix} &= \left[\mathbf{M}^{(k+1)}\right]^{-1}\begin{pmatrix}m_0^{(k+1)}\\ m_1^{(k+1)}\end{pmatrix} = \\ &= \frac{k+1}{k}\left\{\mathbf{M}^{-1}-\frac{1}{k}\mathbf{D}\right\}\left\{\frac{k}{k+1}\begin{pmatrix}m_0\\ m_1\end{pmatrix}+\frac{1}{k+1}\begin{pmatrix}f_k\\ x_kf_k\end{pmatrix}\right\} \\ &= \begin{pmatrix}\beta_k\\ \alpha_k\end{pmatrix}-\frac{1}{k}\mathbf{D}\begin{pmatrix}m_0\\ m_1\end{pmatrix}+\mathbf{M}^{-1}\frac{1}{k}\begin{pmatrix}f_k\\ x_kf_k\end{pmatrix}+\frac{1}{k^2}\mathbf{D}\begin{pmatrix}f_k\\ x_kf_k\end{pmatrix}.\end{aligned}$$

5.2. A pontsorozat korlátossága

Az eljárás konvergenciájának előkészítéseként belátjuk, hogy az SRA_D által előállított $\{x_n\}_{n=k}^{\infty}$ pontsorozat tetszőleges kiindulásul vett S_k halmaz esetén korlátos lesz. A bizonyítás a [De 01a] cikkben megjelent leírás egyszerűsített formában. Az állítást a 12. tételben fogalmazzuk meg, de ennek több részét a soronkövetkező 13., 14. és 15. lemmában bizonyítjuk csak be.

Először az $\{x_n\}_{n=k}^{\infty}$ pontsorozat természetéről teszünk egy megjegyzést. A pontsorozat monoton részsorozatokból áll, a következő értelemben. Azt mondjuk, hogy egy ugrás van az $\{x_n\}$ pontsorozatban, ha egymásutáni pontok a gyök különböző oldalain vannak, tehát ha az x_k és x_{k+1} egymásutáni pontok és $x_k < \Theta < x_{k+1}$ fennáll (vagy $x_k > \Theta$ esetén $x_{k+1} < \Theta$ teljesül). A 10. tétel alapján az $\{x_n\}_{n=k}^{\infty}$ sorozat monoton növekvő és monoton csökkenő részsorozatokból áll, ezeket a részsorozatokat választják el ugrások. Például előfordulhat, hogy $x_{l_1} < x_{l_1+1} < \ldots < x_{l_1+s_1} < \Theta < x_{l_1+s_1+1} = x_{l_2}$ áll fenn, aztán az $x_{l_2} > x_{l_2+1} > \ldots > x_{l_2+s_2} > \Theta > x_{l_2+s_2+1} = x_{l_3}$ egyenlőtlenségek teljesülnek, valamilyen $l_1, s_1, l_2, s_2, \ldots$ nemnegatív egészekre, ahol $s_1, s_2, \ldots$ tetszőleges, $l_1 < l_2 < \ldots$ monoton növő egészek.

Bevezetjük a zárójelező pontpár fogalmát is. Ha léteznek olyan x_i és x_j pontok, amelyekre $x_i < \Theta$ és $\Theta < x_j$ fennáll, akkor az x_i, x_j pontpárt zárójelezőnek nevezzük (ezek a pontok nem okvetlenül egymásutániak). Egy SRA_D által előállított $\{x_n\}_{n=k}^{\infty}$ pontsorozatban vagy van egy zárójelező pontpár, vagy csak egyetlen monoton növő (vagy csökkenő) pontsorozat alkotja az $\{x_n\}_{n=k}^{\infty}$ halmazt. Ha nincs ugrás, és a teljes pontsorozat csak (például) monoton növő pontokból áll, akkor ezt természetesen korlátozza Θ. Ha pedig van ugrás a pontsorozatban, akkor nyilván van zárójelező pontpár is. Ezért a pontsorozat korlátosságának bizonyítását csak a zárójelező pontpár megléte esetén kell elvégezni.

A bizonyítás egyszerűbb leírása céljából tegyük fel, hogy a pontok különbözőek, továbbá az általánosság megszorítása nélkül feltehetjük, hogy az S_k halmazban lévő pontokat sorbarendeztük, vagyis $x_0 < x_1 < \ldots < x_{k-1}$ és létezik egy zárójelező pontpár – ilyen például az (x_0, x_{k-1}). Tegyük még fel, hogy $x_k < \Theta$, a másik eset hasonlóan kezelhető.

12. Tétel. *Legyen* $S_k = \{x_i, f_i\}_{i=0}^{k-1}$ *adott,* $(x_0 < x_1 < \ldots, < x_{k-1})$ *és legyen* (x_0, x_{k-1}) *egy zárójelező pontpár. Legyenek* x_k *és* x_{k+1} *az* SRA_D *által előállított egymásutáni pontok, ahol* $x_k < \Theta$*. Ekkor* x_{k+1} *nem lehet messzebb a* Θ *gyöktől, mint akármelyik előzőleg meghatározott pont, vagyis*

$$|x_{k+1} - \Theta| \leq \tau = \max_{i=0,\ldots,k} |\Theta - x_i|.$$

Bizonyítás. Az $x_k < \Theta$ feltevés és a $x_k < x_{k+1}$ monotonitási tulajdonság miatt az x_{k+1} csak abban az esetben sértheti meg a kimondott korlátosságot, ha Θ-nál nagyobb. Két esetet kell megkülönböztetni:

(I) az első esetben x_k kisebb minden eddigi pontnál $x_k < x_i, i = 0, 1, \ldots, k-1$ (a legkisebb pont esete), illetőleg

(II) az x_k pont a többi pont között van, vagyis $x_r < x_k < x_{r+1}$ áll fenn, valamilyen $0 \leq r \leq k-1$ index esetén (középső pont esete).

Mindkét esetben a feladat a maximális $x_{k+1} - x_k$ lépéshossz meghatározására vezethető vissza, ahol ezt a lépéshosszat, mint f_k függvényét már megadtuk a 10. tételben.

Tekintsük az (I) esetet, feladatunk rögzített $x_0, \ldots, x_k$ és $f_0, \ldots, f_{k-1}$ esetén a $h(f_k)$ maximális értékének meghatározása, ahol $x_k < x_0$ és

$$h(f_k) = x_{k+1} - x_k = -f_k \frac{\sum_{i=0}^{k-1}(x_k - x_i)^2}{\sum_{i=0}^{k-1}\sum_{j=i+1}^{k}(x_j - x_i)(f_j - f_i)},$$

továbbá $f_k \in I_k = (-\infty, f_0)$. Előállítjuk a $h(f_k)$ függvény deriváltját. Mivel

$$\frac{\partial den(k+1)}{\partial f_k} = \frac{\partial \sum_{i=0}^{k-1}\sum_{j=i+1}^{k}(x_j - x_i)(f_j - f_i)}{\partial f_k} = \sum_{i=0}^{k-1}(x_k - x_i),$$

így a $h'(f_k)$ értékét a következő módon lehet meghatározni:

$$\begin{aligned} h'(f_k) = \frac{\partial h(f_k)}{\partial f_k} &= \left[\frac{-f_k \sum_{i=0}^{k-1}(x_k - x_i)^2}{den(k+1)}\right]' \\ &= \frac{-\sum_{i=0}^{k-1}(x_k - x_i)^2 \times den(k+1) + f_k \sum_{i=0}^{k-1}(x_k - x_i)^2 \sum_{i=0}^{k-1}(x_k - x_i)}{[den(k+1)]^2} \qquad (5.11) \\ &= \left[\frac{\sum_{i=0}^{k-1}(x_k - x_i)^2}{(den(k+1))^2}\right]\left[f_k \sum_{i=0}^{k-1}(x_k - x_i) - den(k+1)\right]. \end{aligned}$$

A szorzat első tényezője pozitív, a szögletes zárójelben lévő második tényező pedig

mindig negatív, ahogy azt az alábbi átalakításokkal megmutatjuk.

$$
\begin{aligned}
f_k \sum_{i=0}^{k-1}(x_k - x_i) - den(k+1) &= f_k \sum_{i=0}^{k-1}(x_k - x_i) - \sum_{i=0}^{k-1}\sum_{j=i+1}^{k}(x_j - x_i)(f_j - f_i) \\
&= -\sum_{i=0}^{k-2}\sum_{j=i+1}^{k-1}(x_j - x_i)(f_j - f_i) + \sum_{i=0}^{k-1}(x_k - x_i)f_i \qquad (5.12)\\
&= -den(k) + k(m_0 x_k - m_1) = -k^2(-m_0 M_1 + m_1) + k(m_0 x_k - m_1) \\
&= -k[k(-m_0 M_1 + m_1) - m_0 x_k + m_1)] = -k[(k+1)(-m_0 M_1 + m_1) + m_0 M_1 - m_0 x_k] \\
&= -k[(k+1)(-m_0 M_1 + m_1) + m_0(M_1 - x_k)].
\end{aligned}
$$

Mivel most $x_k < x_i, i = 0, 1, \ldots, k-1$, tehát $x_k < M_1$ triviálisan igaz, a 7. lemma miatt $m_0 > 0$ szintén igaz, így $m_0(M_1 - x_k) > 0$ is igaz. Továbbá $(-m_0 M_1 + m_1) > 0$ a 8. tétel (ii) része miatt áll fenn. Ezek miatt az (5.12) szögletes zárójelben lévő kifejezésének mindkét tagja pozitív, az egész kifejezés negatív, így (5.11) valóban negatív. Összegezve a fentieket a

$$h'(f_k) < 0, \text{ ha } f_k \in I_k, \qquad (5.13)$$

vagyis a $h(f_k)$ függvény a szélsőértékét, a maximális τ^* lépéshosszat a $f_k \to -\infty$ esetben veszi fel; ezt a határértéket a l'Hopital szabállyal lehet meghatározni:

$$\tau^* = \lim_{f_k \to -\infty}\left(-f_k \sum_{i=0}^{k-1}\frac{(x_k - x_i)^2}{den(k+1)}\right) = \frac{\sum_{i=0}^{k-1}(x_k - x_i)^2}{\sum_{i=0}^{k-1}(x_i - x_k)}, \qquad (5.14)$$

Itt az $\{x_n\}$ korlátosságának megmutatásához megint két esetet kell külön megvizsgálni.

(A) eset: Legyen a Θ gyöktől az x_k pont a legmesszebb, vagyis $\tau = \max_{i=0,\ldots,k}|\Theta - x_i|$ az $i = k$ indexre valósul meg. Ekkor a tétel állítása az $x_{k+1} - x_k \le \tau^* \le 2(\Theta - x_k)$ egyenlőtlenség formájában fogalmazható meg.

(B) eset: Legyen a Θ gyöktől az x_{k-1} pont a legmesszebb, ekkor a bizonyítandó egyenlőtlenség a $\tau^* < x_{k-1} - x_k$ formát ölti.

Tekintsük először az (A) esetet. A bizonyítandó $\tau^* < 2(\Theta - x_k)$ egyenlőtlenség a τ^* értékét megadó (5.14) egyenlőség felhasználásával

$$\sum_{i=0}^{k-1}(x_k - x_i)^2 \le 2(\Theta - x_k)\sum_{i=0}^{k-1}(x_i - x_k) \qquad (5.15)$$

formába írható. Vegyük észre, hogy mivel $|x_i - \Theta| \le \Theta - x_k$, vagyis $x_i - x_k \le 2(\Theta - x_k)$ fennáll minden $i = 0, 1, \ldots, k-1$ indexre. Az i-edik egyenlőtlenséget megszorozva az $(x_i - x_k) > 0$ kifejezéssel kapjuk, hogy

$$(x_k - x_i)^2 \le 2(\Theta - x_k)(x_i - x_k), \ i = 0, 1, \ldots, k-1, \qquad (5.16)$$

Ezt a k darab egyenlőtlenséget összeadva éppen a bizonyítandó (5.15) egyenlőtlenséget kapjuk.

A (B) eset hasonlóan kezelhető. Belátandó a $\tau^* < x_{k-1} - x_k$ egyenlőtlenség; ebbe behelyettesítjük a τ^* kifejezését a (5.14) egyenletből és az egyenlőtlenség mindkét oldalát a $\sum_{i=0}^{k-1}(x_i - x_k)$ kifejezéssel szorozva kapjuk, hogy

$$\sum_{i=0}^{k-1}(x_i - x_k)^2 \leq (x_{k-1} - x_k)\sum_{i=0}^{k-1}(x_i - x_k) \tag{5.17}$$

alakú lesz a bizonyítandó egyenlőtlenség. Mivel itt $x_i < x_{k-1}$ a sorbarendezés miatt, így $x_i - x_k \leq x_{k-1} - x_k$ minden $i = 0, 1, \ldots, k-1$ esetén. Megszorozva ezen egyenlőtlenségek mindkét oldalát a $(x_i - x_k) > 0$ kifejezéssel és összeadva ezeket pontosan a bizonyítandó (5.17) egyenlőtlenséget kapjuk. Tehát ezzel elintéztük az (I) esetet, amikor x_k minden eddigi pontnál kisebb volt (akár az (A), akár a (B) eset áll fenn).

A (II) eset, vagyis amikor az x_k pont a már előzőleg számított pontok között van (középső pont esete), jóval hosszabb bizonyítást igényel, de lényegében csak egyetlen egyenlőtlenséget kell belátni.

Legyenek most is az $x_i, i = 0, 1, \ldots, k-1$ pontok sorbarendezve, a bizonyítás egyszerűsítése miatt legyenek ezek mind különbözőek, (x_0, x_{k-1}) egy zárójelező pár: $x_0 < \Theta < x_{k-1}$, $x_k < \Theta$ egy belső pont, vagyis egy r indexre $x_r \leq x_k \leq x_{r+1}$ igaz (az r index rögzített a továbbiakban). Az $[x_0, x_{k-1}]$ intervallumot a középső x_k pont és a Θ gyök három részre bontja, jelölje a három rész hosszát $\tau_1 = x_k - x_0, \tau_2 = \Theta - x_k, \tau_3 = x_{k-1} - \Theta$. Ezeknek a részeknek a segítségével bevezetünk néhány indexhalmazt: az $i = 0, 1, \ldots, k-2$ indexeket három részre osztjuk:

$$\begin{aligned} I_1 &= \{i \mid x_0 \leq x_i < x_k, i = 0, 1, \ldots, k-2\}, \\ I_2 &= \{i \mid x_k < x_i < \Theta, i = 0, 1, \ldots, k-2\}, \\ I_3 &= \{i \mid \Theta < x_i < x_{k-1}, i = 1, \ldots, k-2\}, \end{aligned}$$

itt I_1 nem üres (hiszen legalább az $i = 0$ index itt van), és $I_1 \cup I_2 \cup I_3 \cup \{k-1\} = \{0, 1, \ldots, k-1\}$. Jelölje τ a tétel állítása szerint az x_k pontból megtehető legnagyobb távolságot, tehát $\tau = \tau_2 + \max(x_{k-1} - \Theta, \Theta - x_0)$, ennek az értékére két esetet különböztetünk meg attól függően, hogy a Θ gyöktől az x_0 pont, vagy x_{k-1} van távolabb:

(A) ha $\tau_1 + \tau_2 > \tau_3$, akkor $\tau = \tau_1 + 2\tau_2$ áll fenn, és

(B) ha $\tau_1 + \tau_2 < \tau_3$, akkor $\tau = \tau_2 + \tau_3$ igaz.

A tétel állítása akkor igaz, ha

$$x_{k+1} - x_k \;=\; -f_k \sum_{i=0}^{k-1} \frac{(x_k - x_i)^2}{den(k+1)} \;\leq\; \tau$$

vagy az ezzel ekvivalens módon felírt

$$-f_k \sum_{i=0}^{k-1}(x_k - x_i)^2 \le \tau \sum_{i=0}^{k-1} \sum_{j=i+1}^{k} (x_j - x_i)(f_j - f_i). \tag{5.18}$$

egyenlőtlenség fennáll. Ezt az egyenlőtlenséget alapegyenlőtlenségnek nevezzük. A bizonyítás abból áll, hogy az (5.18) baloldalán álló összeg valamely tagjára (vagy tagjaira) keresünk olyan, a jobboldali összegben álló tagot (vagy tagokat), amely majorálja a baloldali részösszeget. A bizonyítást az alapegyenlőtlenség baloldalon álló i indexek és a jobboldali i, j indexpárok segítségével szétbontjuk három lemmára, amelyeket a mostani tétel után adunk meg.

A lemmák illetőleg a bizonyítások a következő tagokat használják fel az alapegyenlőtlenség két oldaláról:

a 13. lemmában $i \in I_1$ és $i = k-1$ indexű tagok a baloldalon, $i \in I_1, j = k-1$ indexűek a jobboldalon,

a 14. lemmában a baloldalon $i \in I_3$, a jobboldalon pedig az $i = 0, j \in I_3$ indexpárral megadott tagokat vizsgáljuk (amit a jobboldali i és j jelölések felcserélésével írunk fel a lemmában, hogy az összehasonlításokat könnyebben tudjuk elvégezni),

a 15. lemmában a baloldalon $i \in I_2$ indexű tagok, a jobboldalon pedig az $i = k, j \in I_2$, $i = k, j \in I_3$ és az $i = k, j = k-1$ indexpárral megadott tagokat vizsgáljuk (a lemmában itt is felcseréljük a jobboldali i és j indexeket), továbbá itt használni fogunk egy R_{I_1} maradéktagot, amit a 13. lemmában határozunk meg.

Ezek a lemmák a baloldalon minden egyes lehetséges indexet lefednek, a jobboldalon pedig bizonyos tagokat szerepeltettünk az alapegyenlőtlenségből, de egyik tagot sem választottuk kétszer, tehát a lemmák állítása összegezve pontosan az alapegyenlőtlenséget bizonyítja. □

13. Lemma. *Az alapegyenlőtlenség $i \in I_1$ indexeire igaz az egyenlőtlenség:*

$$-f_k \left[(x_{k-1} - x_k)^2 + \sum_{i \in I_1} (x_k - x_i)^2 \right] \le \tau \sum_{i \in I_1} (x_{k-1} - x_i)(f_{k-1} - f_i), \tag{5.19}$$

és ennek az egyenlőtlenségnek a jobboldalából a baloldalt kivonva kapjuk, hogy az R_{I_1} maradékra

$$R_{I_1} = \tau \sum_{i \in I_1} (x_{k-1} - x_i)(f_{k-1} - f_i) + f_k \left[(x_{k-1} - x_k)^2 + \sum_{i \in I_1} (x_k - x_i)^2 \right] \ge -2 f_k \tau_2 \sum_{i \in I_1} (x_k - x_i) > 0.$$

Bizonyítás. Belátjuk az

$$-f_k \left[(x_{k-1} - x_k)^2 + \sum_{i \in I_1} (x_k - x_i)^2 \right] \le \tau \sum_{i \in I_1} (x_{k-1} - x_i)(f_{k-1} - f_i),$$

egyenlőtlenséget. Ebben és az alapegyenlőtlenség bizonyítását szolgáló további két lemmában eljárásunk alapvonása, hogy a bizonyítandó egyenlőtlenség baloldalát növeljük, a

jobboldalát csökkentjük – többször is, és belátjuk, hogy még az így előállított egyenlőtlenség is fennáll, amiből természetesen az eredeti egyenlőtlenség is következik.

Tekintettel arra, hogy $x_i < x_k, i \in I_1$, tehát $f_{k-1} - f_i > -f_k > 0, i \in I_1$, így a függvényértékek elhagyhatók a bizonyítandó egyenlőtlenségből (ezzel csökkentettük (5.19) jobboldalát) és elég belátni, hogy

$$(x_{k-1} - x_k)^2 + \sum_{i \in I_1} (x_k - x_i)^2 < \tau \sum_{i \in I_1} (x_{k-1} - x_i). \tag{5.20}$$

Mivel $x_k - x_i \leq \tau_1, i \in I_1$, ezért az utolsó egyenlőtlenség baloldalát növeljük, ha az

$$(x_{k-1} - x_k)^2 + \sum_{i \in I_1} (x_k - x_i)^2 \leq (x_{k-1} - x_k)^2 + \tau_1 \sum_{i \in I_1} (x_k - x_i)$$

alakba átírjuk. Továbbá mivel $x_{k-1} - x_i = x_{k-1} - x_k + x_k - x_i, \forall i \in I_1$, ezeket az egyenlőségeket összeadva (az (5.20) jobboldalát csökkentjük)

$$\tau \sum_{i \in I_1} (x_{k-1} - x_i) = \tau \left[\sum_{i \in I_1} (x_{k-1} - x_k) + \sum_{i \in I_1} (x_k - x_i) \right] \geq \tau \left[(\tau_2 + \tau_3) + \sum_{i \in I_1} (x_k - x_i) \right],$$

hiszen a $\sum_{i \in I_1}$ legalább egy tagot tartalmaz. Tehát a két utolsó egyenlőtlenség felhasználásával elég belátni, hogy (5.20) helyett fennáll a következő:

$$(x_{k-1} - x_k)^2 + \tau_1 \sum_{i \in I_1} (x_k - x_i) \leq \tau(\tau_2 + \tau_3) + \tau \sum_{i \in I_1} (x_k - x_i). \tag{5.21}$$

Az (A) esetet vizsgáljuk meg először, amikor $\tau = \tau_1 + 2\tau_2$. Ekkor τ és az $(x_{k-1} - x_k)^2 = (\tau_2 + \tau_3)^2$ helyettesítéssel az (5.21) jobboldalából kivonva a baloldalt kapjuk a különbségre, hogy

$$\begin{aligned} &(\tau_1 + 2\tau_2)(\tau_2 + \tau_3) + 2\tau_2 \sum_{i \in I_1} (x_k - x_i) - (\tau_2 + \tau_3)^2 \\ &= \quad 2\tau_2 \sum_{i \in I_1} (x_k - x_i) + (\tau_2 + \tau_3)(\tau_1 + \tau_2 - \tau_3) \geq 2\tau_2 \sum_{i \in I_1} (x_k - x_i), \end{aligned}$$

mivel $\tau_1 + \tau_2 \geq \tau_3$ az (A) eset miatt. A szumma minden egyes tagjában pozitív, tehát a maradék pozitív. Visszaszorozva a $-f_k > 0$ függvényértékkel, kapjuk a R_{I_1} maradék tag lemmában megadott alsó korlátját.

A (B) esetben $\tau = \tau_2 + \tau_3$ és $\tau_1 + \tau_2 < \tau_3$ (vagyis $\tau_3 - \tau_2 \geq \tau_1$) , tehát a bizonyítandó (5.21) egyenlőtlenség jobboldalából kivonva a baloldalt a különbség

$$\begin{aligned} &\tau(\tau_2 + \tau_3) + \tau \sum_{i \in I_1} (x_k - x_i) - (\tau_2 + \tau_3)^2 - \tau_1 \sum_{i \in I_1} (x_k - x_i) \\ &= \quad (\tau_2 + \tau_3 - \tau_1) \sum_{i \in I_1} (x_k - x_i) + (\tau_2 + \tau_3)(\tau - \tau_2 - \tau_3) = 2\tau_2 \sum_{i \in I_1} (x_k - x_i) \end{aligned}$$

(ahol τ_1 helyett a nála nagyobb $\tau_3 - \tau_2$ értéket helyettesítettünk), a különbség egy alsó korlátjára megint a $2\tau_2 \sum_{I_1}(x_k - x_i)$ értéket kapjuk, amit megszorozva az $-f_k > 0$ függvényértékkel kapjuk az R_{I_1} maradéktag keresett korlátját. □

14. Lemma. *Az alapegyenlőtlenség $i \in I_3$ indexeire igaz az alábbi egyenlőtlenség:*

$$-f_k \sum_{i \in I_3} (x_i - x_k)^2 \leq \tau \sum_{i \in I_3} (x_i - x_0)(f_i - f_0). \tag{5.22}$$

Bizonyítás. Most minden $f_i, i \in I_3$ függvényérték pozitív, és ezért $f_i - f_0 \geq -f_k, i \in I_3$ fennáll, tehát a bizonyítandó egyenlőtlenségből a függvényértékek elhagyhatók, ezzel csak csökkentjük a jobboldalt. Elég bebizonyítani, hogy

$$\sum_{i \in I_3} (x_k - x_i)^2 \leq \tau \sum_{i \in I_3} (x_i - x_0).$$

fennáll. Mivel itt $(x_i - x_k) \leq \tau_2 + \tau_3$, ezért a baloldalt megnövelhetjük, valamint az $(x_i - x_0) \geq (x_i - x_k)$ egyenlőtlenség felhasználásával lecsökkentjük a jobboldalt és elég belátni a

$$\sum_{i \in I_3} (x_k - x_i)^2 \leq (\tau_2 + \tau_3) \sum_{i \in I_3} (x_i - x_k) \leq \tau \sum_{i \in I_3} (x_i - x_k) \leq \tau \sum_{i \in I_3} (x_i - x_0). \tag{5.23}$$

egyenlőtlenségekből a középsőt, vagyis bizonyítandó, hogy

$$(\tau_2 + \tau_3) \sum_{i \in I_3} (x_i - x_k) \leq \tau \sum_{i \in I_3} (x_i - x_k). \tag{5.24}$$

Az (A) esetben, amikor $\tau_3 < \tau_1 + \tau_2$ és $\tau = \tau_1 + 2\tau_2$ igaz, akkor a bizonyítandó egyenlőtlenség

$$(\tau_2 + \tau_3) \sum_{i \in I_3} (x_i - x_k) \leq (\tau_1 + 2\tau_2) \sum_{i \in I_3} (x_i - x_k). \tag{5.25}$$

Itt a két összeg azonos már, továbbá $\tau_1 + 2\tau_2 - (\tau_2 + \tau_3) = \tau_1 + \tau_2 - \tau_3 > 0$ igaz, tehát az egyenlőtlenség fennáll.

A (B) esetben, amikor $\tau = \tau_2 + \tau_3$ akkor az (5.24) bizonyítandó egyenlőtlenség alakja

$$(\tau_2 + \tau_3) \sum_{i \in I_3} (x_i - x_k) \leq (\tau_2 + \tau_3) \sum_{i \in I_3} (x_i - x_k) \tag{5.26}$$

amely evidensen igaz. □

15. Lemma. *Az alapegyenlőtlenség baloldalán szereplő $i \in I_2$ tagjaira fennáll, hogy :*

$$-f_k \sum_{i \in I_2} (x_i - x_k)^2 \leq \tau \left[\sum_{i \in I_2} (x_i - x_k)(f_i - f_k) + \sum_{i \in I_3, i = k-1} (f_i - f_k)(x_i - x_k) \right] + R_{I_1}, \tag{5.27}$$

ahol a jobboldalon szereplő $R_{I_1} \geq 2\tau_2(-f_k) \sum_{I_1}(x_k - x_i)$ maradék alsó korlátjának értékét a 13. lemmában határoztuk meg.

Bizonyítás. Két esetet fogunk külön megvizsgálni, ha az $M_1 = (1/k)\sum_{i=0}^{k-1} x_i < x_k$ és ha az $M_1 > x_k$ egyenlőtlenség igaz. Vegyük észre, hogy $M_1 = x_k$ csak akkor következhet be, ha $m_0 = 0, M_1 = x_k = \Theta$, lásd a 7. lemmát.

Tekintsük először az $M_1 < x_k$ esetet. Kiindulunk a $\sum_{i=0}^{k-1}(x_i - M_1) = 0$ egyenlőségből, ami átrendezve az

$$\sum_{x_i > M_1} (x_i - M_1) = \sum_{x_i < M_1} (M_1 - x_i)$$

alakba írható. Ha itt megnöveljük a jobboldalt és lecsökkentjük a baloldalt az alábbi módon:

$$\begin{aligned} \sum_{x_i < M_1} (M_1 - x_i) &\leq \sum_{i \in I_1} (x_k - x_i), \\ \sum_{x_i > M_1} (x_i - M_1) &\geq \sum_{i \in I_2} (x_i - x_k) + \sum_{i \in I_3, i = k-1} (x_i - x_k) \geq \sum_{i \in I_2} (x_i - x_k), \end{aligned}$$

akkor ezek alapján fennáll a következő egyenlőtlenség

$$\sum_{i \in I_2} (x_i - x_k) \leq \sum_{i \in I_1} (x_k - x_i). \tag{5.28}$$

Visszatérünk a lemma egyenlőtlenségéhez. A bizonyítandó (5.27) egyenlőtlenség baloldalát megnöveljük, először $x_i - x_k \leq \tau_2, i \in I_2$ alapján, másodszor pedig az (5.28) alapján:

$$-f_k \sum_{i \in I_2} (x_i - x_k)^2 \leq \tau_2 (-f_k) \sum_{i \in I_2} (x_i - x_k) \leq \tau_2 (-f_k) \sum_{i \in I_1} (x_k - x_i), \tag{5.29}$$

Így elég belátni, hogy az (5.29) jobboldala kisebb, mint az (5.27) jobboldala. De (5.29) jobboldala pontosan a 13. lemmában megadott R_{I_1} maradék alsó korlátjának fele, ami pedig bizonyítja a lemmánkat az $M_1 < x_k$ esetre.

Tekintsük most a második esetet, amikor $M_1 > x_k$. Megjegyezzük, hogy a 7. lemma alapján $m_0 > 0$ is fennáll (egyébként ez az eset nem fordulhatna elő). Ebből következik, hogy

$$\sum_{i \in I_1} f_i + \sum_{i \in I_2} f_i \; + \sum_{i \in I_3, i = k-1} f_i > 0, \text{ valamint } \sum_{i \in I_3, i = k-1} f_i + \sum_{i \in I_2} f_i \; > \; 0, \tag{5.30}$$

mivel $f_i < 0$ az $i \in I_1$ indexek esetén. Visszatérünk az (5.27) bizonyítására, ami helyett most már elég lesz azt kell belátni, hogy

$$-f_k \sum_{i \in I_2} (x_k - x_i)^2 \leq \tau \left[\sum_{i \in I_2} (f_i - f_k)(x_i - x_k) + \sum_{i \in I_3, i = k-1} (f_i - f_k)(x_i - x_k) \right] \tag{5.31}$$

fennáll. Az (5.31) baloldalát ismét megnöveljük az $(x_i - x_k) \leq \tau_2, i \in I_2$ segítségével:

$$-f_k \sum_{i \in I_2} (x_k - x_i)^2 \leq -f_k \tau_2 \sum_{i \in I_2} (x_i - x_k). \tag{5.32}$$

A (5.31) jobboldalát pedig lecsökkentjük, először $-f_k > 0$, $(x_i - x_k) > 0, i \in I_3, i = k-1$ felhasználásával, majd $f_i > 0, x_i - x_k > \tau_2, i \in I_3, i = k-1$ segítségével, illetőleg annak észrevételével, hogy $\tau > \tau_2$, akár az (A), akár a (B) eset áll fenn:

$$\begin{aligned} \tau \left[\sum_{i \in I_2} (f_i - f_k)(x_i - x_k) + \sum_{i \in I_3, i = k-1} (f_i - f_k)(x_i - x_k) \right] \geq \\ \geq \quad \tau \left[\sum_{i \in I_2} (f_i - f_k)(x_i - x_k) + \sum_{i \in I_3, i = k-1} f_i (x_i - x_k) \right] \\ \geq \quad \tau_2 \sum_{i \in I_2} (f_i - f_k)(x_i - x_k) + \tau_2^2 \sum_{i \in I_3, i = k-1} f_i. \end{aligned} \tag{5.33}$$

Az így kapott (5.32) és (5.33) felhasználásával látható, hogy elég belátni az (5.31) egyenlőtlenség teljesülése helyett a következőt:

$$-f_k \tau_2 \sum_{i \in I_2} (x_i - x_k) \leq \tau_2 \sum_{i \in I_2} (f_i - f_k)(x_i - x_k) + \tau_2^2 \sum_{i \in I_3, i = k-1} f_i. \tag{5.34}$$

Ezen legutóbbi, bizonyítandó egyenlőtlenség jobboldalából kivonva a baloldalt és megjegyezve, hogy $-f_k > 0$ és $f_i < 0, i \in I_2$ kapjuk, hogy

$$\begin{aligned} \tau_2 \sum_{i \in I_2} [(f_i - f_k)(x_i - x_k) + f_k (x_i - x_k)] + \tau_2^2 \sum_{i \in I_3, i = k-1} f_i \\ \geq \quad \tau_2 \sum_{i \in I_2} (f_i - f_k + f_k)(x_i - x_k) + \tau_2^2 \sum_{i \in I_3, i = k-1} f_i \\ = \quad \tau_2 \sum_{i \in I_2} f_i (x_i - x_k) + \tau_2^2 \sum_{i \in I_3, i = k-1} f_i \\ \geq \quad \tau_2^2 \sum_{i \in I_2} f_i + \tau_2^2 \sum_{i \in I_3, i = k-1} f_i = \tau_2^2 \left[\sum_{i \in I_2} f_i + \sum_{i \in I_3, i = k-1} f_i \right], \end{aligned}$$

Ez a kifejezés pozitív, mivel a legutolsó, szögletes zárójelben lévő kifejezés pozitív (az (5.30) alapján), tehát az egész különbség pozitív, vagyis az (5.34) egyenlőtlenség fennáll, amivel a lemmában kimondott egyenlőtlenséget is beláttuk. □

Megjegyzés. Vegyük észre, hogy a 13., 14. és 15. lemma bizonyításában lényeges módon kihasználtuk azt a tényt, hogy a x_0, x_{k-1} pontpár zárójelező volt. Tegyük fel, hogy több zárójelező pontpárunk van, legyen $L = \min [\#\{i | i \in I_1\}, \#\{i | i \in I_3\}]$. Ekkor az alapegyenlőtlenség minden zárójelező pontpárra külön belátható, így L szeresen fennáll, tehát

$$|x_{k+1} - \Theta| \leq \frac{1}{L} \max_{i=0,\ldots,k} |\Theta - x_i|$$

igaz lesz. Lényegében véve ez azt jelenti, hogyha közelebb kerül az x_k pont a gyökhöz (és nem egy, hanem L vagy ennél több zárójelező pontpár van), akkor a lehetséges maximális lépés nagysága kisebbé válik, mint az alapegyenlőtlenségben megadott határ, sőt, legfeljebb a határ $\frac{1}{L}$-ed része lesz.

Példa a korlátosságra, ha középső pont van

Ez a példa azt szemlélteti, hogy ha már van egy zárójelező pontpár, akkor az új pont már nem lehet messzebb a gyöktől, mint az eddigi legmesszebbi pont.

Tekintsük az $f(x)$ függvényt, amely a következő pontokban és függvényértékekkel van megadva: $f(-1) = -1, f(0) = 0, f(\varepsilon_1) = 2\varepsilon_1, \varepsilon_1 > 0$, tehát $\Theta = 0$. Legyen az S_2 pont-függvényhalmaz az $(x_0 = -1, f_0 = -1)$ és a $(x_1 = \varepsilon_1, f_1 = 2\varepsilon_1)$ két darab pont-függvény párral megadva, itt ε_1 tetszőleges kicsi, de pozitív értékű rögzített szám. Vegyük észre, hogy $x_0 = -1, x_1 = \varepsilon_1$ már egy zárójelező pár. Megmutatjuk, hogy x_3 a $[-1, +1]$ intervallumon belül marad, akárhogy is vesszük fel az f többi értékét.

Az SRA_D algoritmus az S_2 halmazból előállítja a soronkövetkező $x_2 = -\varepsilon_2$ pontot, ahol $\varepsilon_2 > 0$ egy kis érték. Látható, hogy ha f_2 értéke nullához közeli, akkor az x_3 pont is nagyon közel lesz az origóhoz. Próbáljuk meg f_2 értékét úgy választani, hogy lehetőleg nagyot ugorjon az algoritmus. Legyen f_2 értéke nagyon közel a -1 értékhez, vagyis legyen $f(x_2) = f(-\varepsilon_2) = -1 + \varepsilon_3, \varepsilon_3 > 0$, de ε_3 is kicsi. Ezek után az $S_3 = S_2 \cup (-\varepsilon_2, -1 + \varepsilon_3)$ halmazból az SRA_D meghatározza az x_3 pontot. Próbáljuk meg ε_3 értékét közelíteni a 0 értékhez, az $x_3 \leq 1$ egyenlőtlenség fennáll minden $\varepsilon_3 > 0$ esetén. Tehát mindegy, mennyire „laposnak" vesszük a függvényt az $x_0 = -1$ és az $x_2 = 0$ között, a korlátosság még mindig fennáll.

Ha pedig az x_2 pontot próbáljuk meg az x_0-hoz közelíteni (az x_1 pontban felvett függvényérték változtatásával, akkor jól láthatóan az x_3 pont még a gyöknél sem lehet nagyobb.

5.3. Az SRA_D konvergenciája

Ebben a részben az SRA_D által előállított pontsorozat konvergenciáját látjuk be a determinisztikus függvényértékek esetére.

16. Tétel. *Tegyük fel, hogy az $f(x)$ függvényre fennáll az $A(f-1)$ feltevés. Egy tetszőleges S_k kiindulási halmazból az SRA_D által előállított $\{x_n\}_{n=k}^{\infty}$ pontsorozat a Θ gyökhöz konvergál.*

Bizonyítás. Az előző 5.2 részben beláttuk, hogy az SRA_D által előállított $\{x_n\}_{n=k}^{\infty}$ pontsorozat korlátos, jelölje L és U a pontsorozat egy alsó, illetőleg egy felső korlátját. A

sorozatnak legalább egy torlódási pontja van. Belátjuk, hogy csak egy torlódási pontja van, és ez éppen a Θ gyök. Azt mondtuk, hogy a pontsorozatban ugrás van, ha két egymásutáni pontot a gyök elválaszt: például $x_k < \Theta < x_{k+1}$. Két esetet kell megkülönböztetni:

(i) Tegyük fel, hogy csak véges sok ugrás van a $\{x_n\}_{n=k}^{\infty}$ pontsorozatban, az utolsó ugrás utáni pont legyen x_N. Ekkor a későbbi $x_{N+1}, x_{N+2}, \ldots$ pontok egy monoton növekvő (vagy csökkenő) sorozatot alkotnak. Be kell látni, hogy ez a monoton sorozat konvergál a Θ gyökhöz. A bizonyítást indirekt módon végezzük. Tegyük fel, hogy $x_N < \Theta$ és létezik egy $y \neq \Theta$ konvergenciapont, vagyis $\lim_{n\to\infty} x_{N+n} \to y$, ahol $f(y) < 0$ is teljesül. Ha $x_{N+n} \to y$, akkor $M_1 \to y, m_0 \to f(y), \alpha_k \to \overline{\alpha}_k$, amelyre $\overline{\alpha}_k \geq \delta_L > 0$. Így a 7. lemma alapján kapott kifejezésből

$$x_k = M_1 - \frac{m_0}{\alpha_k} \to y - \frac{f(y)}{\overline{\alpha}_k} \neq y, \text{ha } k \to \infty,$$

ugyanis $-f(y)/\overline{\alpha}_k > 0$ a fentiek miatt. Tehát ellentmondást kaptunk, egyetlen $y \neq \Theta$ sem lehet konvergenciapont.

(ii) Tegyük fel, hogy végtelen sok ugrás van a $\{x_n\}_{n=k}^{\infty}$ pontsorozatban. Először belátjuk, hogy ebben az esetben a lépéshossz nullához tart, vagyis tetszőleges kis $\varepsilon > 0$ esetén található N, amelyre $|x_{N+i} - x_{N+i+1}| \leq \varepsilon, i = 1, 2, \ldots$ fennáll. Mivel ekkor végtelen sok pont van mind a $[L, \Theta)$, mind a $(\Theta, U]$ intervallumokban, mindkét intervallumban van egy torlódási pont.

Tegyük fel, hogy két különböző T_1 és T_2 torlódási pontunk van, $T_1 < \Theta, T_2 > \Theta$. Legyen $\delta = \min(\Theta - T_1, T_2 - \Theta)$, amely egy pozitív távolság a feltevés miatt. Tekintsük a T_1, T_2 pontok egy-egy $0 < \varepsilon < \delta/2$ sugarú környezetét, és legyen az $\{x_n\}_{n=0}^{N}$ sorozatban található pontok közül ezen környezetekben lévő pontok száma $K_1(N)$ és $K_2(N)$:

$$K_1(N) = \#\{x_i \mid |x_i - T_1| < \varepsilon, i = 0, 1, \ldots, N\},$$
$$K_2(N) = \#\{x_i \mid |x_i - T_2| < \varepsilon, i = 0, 1, \ldots, N\},$$

és jelölje $N_1(N)$ és $N_2(N)$ a gyöknél kisebb, illetőleg nagyobb pontok számát:

$$N_1(N) = \#\{x_i \mid x_i < \Theta, i = 0, 1, \ldots, N\},$$
$$N_2(N) = \#\{x_i \mid x_i > \Theta, i = 0, 1, \ldots, N\}.$$

Vegyük észre, hogy $N_1(N) + N_2(N) = N$ fennáll, hiszen egyetlen x_i pont sem lehet Θ-val egyenlő (lásd a 10. tétel utáni következményt). Nyilván $K_i(N) \to \infty, i = 1, 2$ ha $N \to \infty$, mivel mindkét T_1, T_2 pont torlódási pont. Felhasználva a 8. tétel (v) részét az $x_{k+1} - x_k = -f_k \sum_{i=0}^{k-1} \frac{(x_k - x_i)^2}{\sum_{i=0}^{k-1} \sum_{j=i+1}^{k} (x_j - x_i)(f_j - f_i)}$ lépéshossz értékét átírhatjuk:

$$
\begin{aligned}
h(f_N) &= -f_N \frac{\sum_{i=0}^{N-1}(x_N - x_i)^2}{\sum_{i=0}^{N-1}\sum_{j=i+1}^{N}(x_j - x_i)(f_j - f_i)} \\
&= \frac{-f_N}{\sum_{i=0}^{N-1}\sum_{j=i+1}^{N}\frac{f_j - f_i)}{(x_j - x_i)}\,\frac{(x_j - x_i)^2}{\sum_{i=0}^{N-1}\sum_{j=i+1}^{N}(x_j - x_i)^2}}\;\frac{\sum_{i=0}^{N-1}(x_N - x_i)^2}{\sum_{i=0}^{N-1}\sum_{j=i+1}^{N}(x_j - x_i)^2} \qquad (5.35)\\
&= \frac{-f_N}{\sum_{i=0}^{N-1}\sum_{j=i+1}^{N}\alpha_{ij}\lambda_{ij}^{(N)}}\;\frac{\sum_{i=0}^{N-1}(x_N - x_i)^2}{\sum_{i=0}^{N-1}\sum_{j=i+1}^{N}(x_j - x_i)^2}
\end{aligned}
$$

Az első tört korlátos, mivel α_{ij} korlátosak, és f_N is korlátos, hiszen az $\{x_n\}_{n=0}^{\infty}$ pontsorozat is korlátos. Legyen C egy, az első törtnél nagyobb konstans, és legyen $\Delta = U - L$ a pontokat tartalmazó korlátos intervallum hossza, ekkor a lépéshossz majorálható

$$
h(f_N) \le C\,\frac{\sum_{i=0}^{N-1}(x_N - x_i)^2}{\sum_{i=0}^{N-1}\sum_{j=i+1}^{N}(x_j - x_i)^2} \le \frac{CN\Delta^2}{\sum_{i=0}^{N-1}\sum_{j=i+1}^{N}(x_j - x_i)^2} \qquad (5.36)
$$

Mivel itt $(x_j - x_i)^2 \ge (\delta - \varepsilon)^2$ ha x_i és x_j a gyök különböző oldalain vannak, tehát a $K(N) = \min(K_1(N), K_2(N))$ jelölést használva felírhatjuk, hogy

$$
\begin{aligned}
&\sum_{i=0}^{N-1}\sum_{j=i+1}^{N}(x_j - x_i)^2 \ge \\
&\quad\ge \frac{1}{2}\left[N_1(N)K_2(N)(\delta-\varepsilon)^2 + N_2(N)K_1(N)(\delta-\varepsilon)^2\right] \\
&\quad\ge \frac{1}{2}\left[K(N)(\delta-\varepsilon)^2[N_1(N)+N_2(N)]\right] = K(N)(\delta-\varepsilon)^2 N/2 \qquad (5.37)
\end{aligned}
$$

tehát egy felső korlát adható az (5.36) egyenletben szereplő lépéshosszra:

$$
h(f_N) \le \frac{2CN\Delta^2}{K(N)(\delta-\varepsilon)^2 N} = \frac{2C\Delta^2}{(\delta-\varepsilon)^2}\frac{1}{K(N)}. \qquad (5.38)
$$

A jobboldal 0-hoz tart, mivel $K(N) \to \infty$, ha $N \to \infty$, – ugyanis T_1, T_2 pontokról feltettük, hogy torlódási pontok. Tehát a lépéshossz konvergál 0-hoz.

Ez pedig (végtelen sok ugrás és két különböző torlódási pont létezése esetén) ellentmondásra vezetett, hiszen ha az algoritmus egy ugrást hajt végre, akkor a lépéshossznak legalább $|T_1 - \Theta| - \varepsilon = \delta - \varepsilon > \varepsilon > 0$ nagynak kell lennie ahhoz, hogy a másik torlódási pont ε környezetét el tudja érni.

Ugyanezeket a meggondolásokat lehet használni, ha az egyik torlódási pont a gyök, ha például $\Theta = T_2$ és $T_1 < \Theta$ igaz: a megfelelő környezetbe nem tud átugrani az algoritmus. Tehát mindkét oldali pontsorozatnak csak Θ-val azonos torlódási pontja lehet. □

Megjegyzések.

Az eljárás előnye, hogy a függvényről nem kell feltenni a differenciálhatóságot, szakaszonként lineáris függvények esetén is használható, a paraméterek felfrissítése önmagában is egyszerű, a javasolt mátrix-inverz felfrissítés pedig szokásos és – ami fontosnak tűnik, – a többdimenziós esetre is átvihető. Az algoritmus fontos tulajdonsága, hogy zajos függvényértékek esetén is konvergál, amint azt a [De 98d] cikkben szemléltettük (más jól ismert egydimenziós gyökkereső eljárások, mint például a Newton módszer, általában nem konvergálnak zajos függvények esetén).

A konvergencia a kapott numerikus eredmények alapján elég lassú. További kellemetlen tulajdonsága az algoritmusnak, hogy nem monoton, vagyis ha x_k, x_{k+1} egymásutáni pontok, akkor x_{k+1} nincs mindig közelebb a Θ gyökhöz, mint x_k, sőt az M_1 átlag sem monoton konvergáló.

5.4. A zajos függvény esete

A szukcesszív regressziós approximációk módszerét némiképpen módosítjuk a zajos függvényértékek esetére. Az eljárás egyszerűsítése céljából feltesszük, hogy az algoritmus által előállított pontsorozatot egy $[L, U]$ intervallumra csonkoljuk, amely tartalmazza a valódi gyököt – $\Theta \in [L, U]$.

$A(x-2)$ Feltesszük, hogy ha az SRA_S egy olyan x_k közelítő gyököt állít elő, amely kisebb L-nél (vagy nagyobb U-nál), akkor a gyök értékét a határponttal tesszük egyenlővé, legyen $x_k = L$ (vagy $x_k = U$).

Ezt a feltevést az alábbi SRA_S algoritmus 2. lépésébe építettük be. Tegyük fel továbbá, hogy az $f(x_i)$ függvényértékeket nem lehet pontosan kiszámítani; a pontos érték helyett csak a függvényérték és egy additív hiba összege áll rendelkezésünkre. Ez két okból fordulhat elő: a.) a függvény („megfigyelés") természete önmagában véletlen, vagy pedig b.) a függvényérték kiszámítására használt eljárás (például Monte Carlo integrálás) egy véletlen hibával terhelt értéket tud csak előállítani. Ezeket az eseteket összefoglalóan a zajos függvény esetének nevezzük.

Ezen zajos függvény feltevés esetén az $x_k, x_{k+1}, \ldots, \alpha_k, \beta_k$ mennyiségek valószínűségi változók lesznek. A valószínűségi változókat egy hullám ($\widetilde{\ }$) jellel különböztetjük meg a determinisztikus megfelelőiktől (ha ez nem okoz zavart, akkor lehagyjuk a hullám jelölést).

Így most az $f_i = f(x_i)$ pontos függvényérték helyett csak az $\widetilde{f}_i = f_i + \varepsilon_i$ áll a rendelkezésünkre ahol ε_i egy véletlen zaj. Erre a következő feltevéssel élünk:

$A(\varepsilon-3)$ Legyenek az $\varepsilon_i, i = 0, 1, \ldots$ azonos eloszlású, teljesen független valószínűségi változók, amelyekre $E(\varepsilon_i) = 0$, $D^2(\varepsilon_i) = \sigma^2, i = 0, 1, \ldots, E(\varepsilon_i \varepsilon_j) = 0, i, j = 0, 1, \ldots, i \neq j$.

Az $\varepsilon_i, i = 0, 1, \ldots, k-1$ valószínűségi változók által generált legkisebb σ-algebra $\mathcal{A}_{k-1}$, ahol $\mathcal{A}_i \subset \mathcal{A}_{i+1}, i = 2, 3, \ldots$, a $\widetilde{S}_k = \{\widetilde{x}_i, \widetilde{f}_i\}_{i=0}^{k-1}$, $\widetilde{x}_k, \widetilde{m}_0$, valószínűségi változók az $\mathcal{A}_{k-1}$ σ-algebrán mérhetőek. Jegyezzük meg, hogy $\widetilde{x}_k$ az $\widetilde{S}_k = (\varepsilon_0, \ldots, \varepsilon_{k-1}, \widetilde{x}_0, \ldots, \widetilde{x}_{k-1})$-tól függ, $\widetilde{x}_k \in \mathcal{A}_{k-1}$, de $\widetilde{x}_k$ független az $\varepsilon_k \in \mathcal{A}_k$-tól.

5.4.1. Az SRA_S algoritmus

Módosítjuk az SRA_D algoritmust a zajos függvényértékek esetére, hogy az egyetlen lehetséges szingularitási pontot elkerüljük. Addig nem számítunk ki új pontot, amíg $\widetilde{\alpha}_k \le \delta_L/2$ vagy $\widetilde{\alpha}_k \ge 2\delta_U$ teljesül, hanem ilyenkor a legutolsó $\widetilde{x}_k$ pontban többször kiszámítjuk a függvényértéket:

SRA_S - egydimenziós gyökkeresés, zajos függvény)

0. Tegyük fel, hogy rendelkezésünkre áll egy kiindulási $\widetilde{S}_k = \{x_i, \widetilde{f}_i\}_{i=0}^{k-1}$ halmaz és legyen a k iterációs számláló az adott pontok száma.
1. Számítsuk ki a $\widetilde{g}_k(x) = \widetilde{\alpha}_k x + \widetilde{\beta}_k$ függvény paramétereit az $\widetilde{S}_k$-ból. Ha $\delta_L/2 \le \widetilde{\alpha}_k$ és $\widetilde{\alpha}_k \le 2\delta_U$ fennáll, akkor menjünk a 2. lépésre, egyébként legyen $\widetilde{x}_k = \widetilde{x}_{k-1}$, számítsuk ki az $\widetilde{f}_k = \widetilde{f}(\widetilde{x}_k)$ értéket, a $\widetilde{x}_k, \widetilde{f}_k$ párt tegyük hozzá $\widetilde{S}_k$-hoz, $\widetilde{S}_{k+1} = \widetilde{S}_k \cup \{\widetilde{x}_k, \widetilde{f}_k\}$, növeljük meg az iterációs számlálót és menjünk vissza az 1. lépés elejére.
2. Határozzuk meg az $\widetilde{x}_k$ közelítő gyököt az $\widetilde{g}_k(x) = 0$ egyenletből. Ha $\widetilde{x}_k < L$, akkor legyen $\widetilde{x}_k = L$, ha $\widetilde{x}_k > U$, akkor legyen $\widetilde{x}_k = U$.
3. Ha $\widetilde{x}_k$ „elég jó", akkor STOP. Egyébként számítsuk ki az $\widetilde{f}(\widetilde{x}_k)$ függvényértéket és legyen $\widetilde{S}_{k+1} = \widetilde{S}_k \cup \{\widetilde{x}_k, \widetilde{f}_k\}$, továbbá $k = k+1$, és menjünk vissza az 1. lépésre.

Vegyük észre, hogy az $\widetilde{S}_k$-ból számított $\widetilde{\alpha}_k$ mennyiségre $P\{\delta_L \le \widetilde{\alpha}_k\} > 0$ és $P\{\widetilde{\alpha}_k \le \delta_U\} > 0$, mivel $0 < \delta_L \le E(\widetilde{\alpha}_k) \le \delta_U < \infty$ fennáll. Tehát az 1. lépésen belüli ciklust csak véges sokszor ismételjük meg (a Borel-Cantelli lemma alapján) 1 valószínűséggel. Tehát $P\{\delta_L/2 \le \widetilde{\alpha}_k\} = P\{\widetilde{\alpha}_k \le 2\delta_U\} = 1$ az SRA_S-el számított $\widetilde{\alpha}_k$ mennyiségre.

5.4.2. A sztochasztikus approximáció

Az alábbiakban az SRA_S algoritmus és a sztochasztikus approximáció néven ismert eljárás közötti összefüggéssel foglalkozunk és megmutatjuk, hogy ezek hasonlóak. Nevezetesen a szukcesszív regressziós approximáció rekurzív képletében a függvényértékek szorzója ugyan valószínűségi változó, de erre bizonyos korlátokat lehet adni.

A sztochasztikus approximáció eljárását Robbins és Monro [RM 51] javasolták egydimenziós gyökkeresésre. Ezt később sok másmilyen esetre kiterjesztették (csak két átfogó irodalmi hivatkozást adunk meg, Kushner és Clark [KC 78], valamint Benveniste et al. [BMP 90] könyvét), itt csak az alapeljárást ismertetjük.

Tekintsük feladatunknak az $f(x) = 0$ egyenlet megoldását, ahol az $f(x)$ pontos függvényérték helyett csak a zajos $[f(x)+\varepsilon]$ érték áll rendelkezésünkre, ahol $E(\varepsilon) = 0, D^2(\varepsilon) = \sigma^2$. A sztochasztikus approximációban az

$$x_{n+1} = x_n - a_n[f(x_n) + \varepsilon_n]. \tag{5.39}$$

alakú rekurziót alkalmazzák a gyök meghatározására (ε_n teljesen független, azonos eloszlású valószínűségi változók, nulla várható értékkel), az így előállított $\{x_n\}$ sorozatot RM sorozatnak nevezzük. Ekkor igaz a Dvoretzky által bizonyított következő tétel (lásd [Dvo 56] p. 50):

17. Tétel. *Tegyük fel, hogy*
(i) rendelkezésünkre áll az $f(x)+\varepsilon$ érték, ahol $E(\varepsilon)=0, D^2(\varepsilon)<\sigma^2$,
(ii) az $\{a_n\}$ pozitív tagokból álló sorozatra $\sum_{n=1}^{\infty} a_n=\infty, \sum_{n=1}^{\infty} a_n^2<\infty$ fennáll,
(iii) valamilyen A, B állandókkal fennáll $|f(x)|<A|x|+B<\infty$,
(iv) minden k-ra $\inf_{1/k<x-\Theta<k} f(x)>0,\ \sup_{1/k<\Theta-x<k} f(x)<0$ igaz.
Ekkor az RM sorozat 1 valószínűséggel konvergál a Θ gyökhöz.

Az általunk adott $A(f-1)$ feltétel erősebb a tételben adott simasági feltételnél, a zajra azonos feltételt tettünk. Vegyük észre, hogy az un. $1/n$ típusú sorozatok kielégítik a feltételeket, vagyis ha valamilyen C', C'' pozitív konstansokkal igaz, hogy $C'/n \le a_n \le C''/n$ fennáll, akkor az a_n együtthatókra tett (ii) feltétel és ezzel a konvergencia teljesül. Megmutatjuk, hogy az általunk lépéshossznak nevezett mennyiség majdnem $1/n$ típusú sorozat.

A Robbins-Monro eljárásban az a_n értékét arra használják, hogy nem túl gyorsan, de csökkentsék a változás mértékét, mintegy mesterségesen csillapítják a véletlenből következő ingadozásokat.

Az SRA eljárást egy automatikus, természetes SA eljárásnak is tekinthetjük. Automatikusnak nevezhetjük, mivel itt az a_n együtthatónak megfelelő lépéshossz sorozatot nem előre adjuk meg, hanem az algoritmus maga határozza meg, az eddigi pontokból és az eddigi függvényértékekből. Természetes sztochasztikus approximációs eljárásnak is nevezhetjük, hiszen az átlagolás gyakran használt eszköz a véletlen ingadozások (zajos függvények) esetén, ilyen például Ruszczynski [Rus 80], vagy Polyak [PJ 92] munkája az optimalizálás területén – a szukcesszív regressziós közelítéseket pedig átlagok határozzák meg (lásd az egydimenziós esetben az m_0, M_1, stb. mennyiségek definícióját).

5.4.3. Lépéshossz az SRA_S algoritmusban

Tetszőleges korlátos és konvergens $\{x_k\}$ pontsorozatokat vizsgálunk a következőkben és alsó korlátot határozunk meg az ezen pontsorozatok által előállított lépéshosszak sorozatára. Tekintsük az SRA algoritmus által megadott rekurzív formulát:

$$x_{k+1} = x_k - f_k \frac{\sum_{i=0}^{k-1}(x_k - x_i)^2}{\sum_{i=0}^{k-1}\sum_{j=i+1}^{k}(x_j - x_i)(f_j - f_i)}.$$

Az f_k függvényérték együtthatóját jelölje l_k, ez felel meg az RM eljárás (5.39) rekurzív képletében a függvényértékek együtthatójának, és ezt nevezzük lépéshossznak:

$$l_k = \frac{\sum_{i=0}^{k-1}(x_k - x_i)^2}{\sum_{i=0}^{k-1}\sum_{j=i+1}^{k}(x_j - x_i)(f_j - f_i)}. \tag{5.40}$$

Vezessük be a $\varphi_k = \alpha_k l_k$ jelölést, vagyis legyen

$$\varphi_k = \frac{\sum_{i=0}^{k-1}(x_k - x_i)^2}{\sum_{i=0}^{k-1}\sum_{j=i+1}^{k}(x_j - x_i)^2}. \tag{5.41}$$

Mivel $l_k = \frac{\varphi_k}{\alpha_k}$ és az SRA_S algoritmus tulajdonságai alapján tudjuk, hogy $0 < \delta_L/2 < \alpha_k < 2\delta_U < \infty$ fennáll, ezért

$$\frac{1}{2\delta_U}\varphi_k \le l_k \le \frac{1}{\delta_L/2}\varphi_k.$$

Tehát ha a φ_k mennyiségre korlátokat tudunk adni, akkor ebből automatikusan korlátokat kapunk a lépéshosszra is. Rögtön látható, hogy $\varphi_k \le 1$, mivel φ_k-nak az (5.41) egyenletben adott számlálója része a nevezőjének. A következő szakaszban a konvergens sorozatokat vizsgáljuk.

5.4.4. Konvergens pontsorozat esete

Tegyük fel, hogy az $\{x_k\}$ pontsorozat konvergál egy x^* ponthoz. Belátjuk, hogy az $\{x_k\}$ pontsorozat által meghatározott (generált) $\{\varphi_k\}$ együtthatók sorozatát alulról korlátozza egy $1/k$ típusú sorozat.

18. Lemma. *Legyen $\{x_k\}$ egy pontsorozat, amelyre $\lim_{k\to\infty} x_k = x^*$, és tegyük fel, hogy $\{x_k\}$ generálja a φ_k sorozatot. Ekkor egy elég nagy k index esetén fennáll, hogy*

$$\frac{1}{9k} \le \varphi_k \le 1.$$

Bizonyítás. A φ_k alsó korlátját a következőképpen határozzuk meg. Tegyük fel, hogy a k index olyan nagy, hogy az $x_k, x_{k+1}, x_{k+2}, \ldots$ pontok mindegyike benne van az x^* pont körüli, δ sugarú környezetben, tehát csak véges számú pont van ezen kívül. Indexeljük át az $x_i, i = 0, 1, \ldots, k-1$ pontsorozatot úgy, hogy x_0 van a legmesszebb az x^* ponttól, x_1 a második legtávolabbi pont, stb. és legyen $\delta \le |x_0 - x^*|/4$ egy rögzített sugár. Megmutatjuk, hogy a φ_k együttható (5.41) adott kifejezésében lévő számláló egy $(x_k - x_i)^2$ tagjának $9k$-szorosa nagyobb, mint k darab (vagy annál kevesebb) nevezőbeli $(x_i - x_j)^2, j = i+1, \ldots, k$ tag vagyis belátjuk a

$$9k\sum_{i=0}^{k-1}(x_k - x_i)^2 \ge \sum_{i=0}^{k-1}\sum_{j=i+1}^{k}(x_j - x_i)^2 \tag{5.42}$$

egyenlőtlenséget. Ezt úgy látjuk be, hogy minden i indexre belátjuk a baloldalon és a jobboldalon lévő i indexű mennyiségek között ezt az egyenlőtlenséget. Tekintsük az $i = 0$ index esetét, ekkor

$$9k(x_k - x_0)^2 \geq (x_0 - x_1)^2 + (x_0 - x_2)^2 + \ldots + (x_0 - x_{k-1})^2 + (x_0 - x_k)^2, \qquad (5.43)$$

fennáll, mivel a jobboldalon álló k darab tag mindegyike kisebb, mint $9(x_0 - x_k)^2$. Például a (5.43) jobboldalán álló első tagra ($j = i + 1 = 1$) írhatjuk, hogy

$$|x_0 - x_1| \leq |x_0 - x^*| + |x^* - x_1| \leq 2|x_0 - x^*| \leq 2[|x_0 - x_k| + |x_k - x^*|] \leq \\ \leq 2[|x_0 - x_k| + \delta] < 3|x_0 - x_k|.$$

A (5.43) jobboldalának a j-edik tagjára ($j = 2, 3, \ldots, k$) kapjuk, hogy

$$|x_0 - x_j| \leq |x_0 - x^*| + |x^* - x_j| \leq 2|x_0 - x^*| \leq 2[|x_0 - x_k| + |x_k - x^*|] \leq 3|x_0 - x_k|.$$

amivel az (5.43) egyenlőtlenséget teljesen beláttuk (az $i = 0$ esetet elintéztük). Hasonlóképpen járunk el a többi i indexre: az (5.42) baloldali i-edik tagjára ($i = 1, 2, \ldots, k-1$) és a jobboldali i indexű tagokra fennáll a

$$9k(x_k - x_i)^2 \geq (x_i - x_{i+1})^2 + (x_i - x_{i+2})^2 + \ldots + (x_i - x_{k-1})^2 + (x_i - x_k)^2,$$

egyenlőtlenség, mivel az itt szereplő $x_i, x_{i+1}, \ldots x_{k-1}$ pontok közül az x_i van a legmesszebb az x^* ponttól a sorbarendezés miatt, így a fenti gondolatmenet alkalmazható. Megjegyezzük, hogy az i-edik egyenlőtlenség jobboldalon csak $k - i$ tag van, tehát a két oldal közötti különbség nő az i növekedésével. Összeadva az így $i = 0, 1, \ldots, k-1$ esetére kapott egyenlőtlenségeket pontosan a belátandó (5.42) egyenlőtlenséget kapjuk, amely átrendezés után a $\{\varphi_k\}$ sorozat keresett alsó korlátját adja.

A felső korlát a φ_k definíciója után tett megjegyzésünk alapján fennáll. □

5.5. Számítógépes tapasztalatok

Tekintsük a következő egydimenziós függvényt:

$$f(x) = \Phi(\mathbf{x}_k + x(\mathbf{d}_k - \mathbf{x}_k)) - p = 0 \qquad (5.44)$$

ahol p egy adott megbízhatósági szint, $0 < p < 1$, az $\mathbf{x}_k, \mathbf{d}_k$ adott vektorok, Φ pedig az n dimenziós normális eloszlás eloszlásfüggvénye. Ez a feladat gyakran előfordul valószínűséggel korlátozott sztochasztikus programozási feladatokban, hiszen ez nem más, mint a $\mathbf{d}_k$ irányú egyenesnek a megengedett megoldások halmaza $\{\mathbf{x}|p = \Phi(\mathbf{x})\}$ burkoló felületével való metszéspontjának meghatározása.

Az SRA_S algoritmust most ennek az egyenletnek a megoldására használjuk, valamint kipróbáljuk iránymenti deriváltak és a gradiens kiszámítására is. A kifejlesztett FORTRAN nyelvű programrendszer részleteit, a példák leírását és a számítógépes futások

példa	becslés	k	x_r	$f(x_r)$	σ_f	σ_1	idő
1	lin.	31015	-0.028815	0.000000	0.000014	0.0015	1.85 sec
	kvad.	34506	-0.028795	-0.000025	0.000014	0.0015	2.07 sec
2	lin.	13749	0.330353	-0.000004	0.000014	0.0018	0.82 sec
	kvad.	35651	0.330361	0.000001	0.000014	0.0017	2.14 sec
3	lin.	16625	0.020190	0.000010	0.000012	0.0014	1.00 sec
	kvad.	19885	0.020231	-0.000031	0.000012	0.0012	1.20 sec
4	lin.	20097	1.79617	0.000032	0.000014	0.0016	1.21 sec
5	lin.	31356	-1.19462	-0.000005	0.000014	0.0015	1.84 sec
	kvad.	29938	-1.19459	-0.000015	0.000014	0.0017	1.77 sec
6	lin.	13417	2.74267	-0.000001	0.000014	0.0016	0.80 sec
	kvad.	18115	2.74262	0.000015	0.000014	0.0013	1.09 sec

5.1. táblázat. $n = 2$ dimenziós gyökkeresés.

becslés	k	x_r	$f(x_r)$	σ_f	σ_1	idő
lin.	10	15.010	0.000347	0.00011	0.0012	0.23 sec
kvad.	10	14.998	-0.000555	0.00010	0.0015	0.23 sec

5.2. táblázat. $n = 10$ dimenziós gyökkeresés.

eredményeit a [De 01b] cikkben közöltük, itt csak a numerikus eredményekből adunk egy rövid összefoglalást.

A számítógépes futások leírása előtt néhány általános megjegyzést teszünk. Az $X = \{\mathbf{x}|\Phi(\mathbf{x}) \geq p\}$ egy konvex halmaz [Pr 95], így egy tetszőleges egyenes esetén három lehetséges eset valamelyike fordul elő. Az (i) esetben az egyenesnek a $\{\mathbf{x}|\Phi(\mathbf{x}) - p = 0\}$ felülettel két metszéspontja van (két gyöke van az egyenletnek), a második esetén (ii) csak egy metszéspont (egy gyök) van, és végül előfordulhat, hogy (iii) nincs metszéspont.

A gyökkereső eljárás hatékonysága függ a kezdeti intervallum (illetőleg az egyenes) megadásától. Most csak az (ii) esetekkel foglalkozunk, amikor van az egyenesen két olyan pont, amelyek közrefogják a gyököt, hiszen ez csak azt jelenti, hogy kell egy pont, amely a megengedett megoldások halmazában belül és egy, amely ezen kívül van. Ilyen pontok a szokásos nemlineáris optimalizálási algoritmusok használata során vagy előfordulnak, vagy könnyen előállíthatók.

A gyökkeresést kétféle becsléssel végeztük el, egy lineáris alakú regresszióval és egy $c_2x^2 + c_1x + c_0$ kvadratikus formájúval. A számítógépes futások alapján megállapíthatjuk, hogy hatékonyságuk nem különbözik szignifikánsan, így nem érdemes kvadratikus közelítést használni.

becslés	k	x_r	$f(x_r)$	σ_f	σ_1	idő
lin.	10	33.082	-0.00029	0.000092	0.0009	0.63 sec
kvad.	10	33.099	0.00022	0.000102	0.0015	0.63 sec

5.3. táblázat. $n = 15$ dimenziós gyökkeresés.

Az 5.1 táblázatban egy kétdimenziós normális eloszlás esetén kapott gyökkeresési eljárás eredményeit mutatjuk be – a hat példa részletes ismertetése a [De 01b] cikkben található. Az 5.2 és 5.3 táblázatban pedig egy $n = 10$, illetőleg $n = 15$ dimenziós normális eloszlásfüggvény esetére mutatunk két példát.

Az 5.1 táblázat első oszlopában a példa sorszámát adjuk meg, aztán a becslés típusát (lineáris vagy kvadratikus), k a pontok és a függvénykiszámítások száma, x_r a végeredményként kapott közelítő gyök, $f(x_r)$ az ezen a helyen kiszámított függvényérték, σ_f az $f(x_r)$ függvényérték szórása (a mintából számítva), σ_1 pedig egy függvénykiszámítás szórása. (Vegyük észre, hogy itt is teljesül a szóráscsökkenésre a 6. fejezetben leírt sejtésünk – az 5.1 táblában egy kiszámított függvényérték szórása 10^{-3}, a végeredmény szórása 10^{-5} körüli.) A numerikus eredményeket összefoglalva mondhatjuk, hogy négy tizedesre pontos gyököt meg lehet határozni egy másodpercnél rövidebb idő alatt, 10 dimenzióig.

Az 5.3 táblázatban használt 15 dimenziós feladat esetére leírunk még egy tapasztalatot. A gradienst úgy becsültük meg, hogy minden koordinátatengelyen 20 pontot vettünk fel, és az ezek segítségével kapott regresszióból határoztuk meg a közelítő gradienst a gyökként kapott, x_r által meghatározott pontban, ez pedig a következő vektor volt: $\nabla\Phi(\cdot) = (0.0, 0.201, -0.016, 0.0, 0.0, 0.0, 0.0, -0.016, -0.022, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0)$, ahol 0.0 egy 0.01-nél kisebb érték volt. Ez a helyzet tipikusnak mondható, az optimalizálás folyamán előforduló gradiensek komponenseinek nagy része közel 0-val egyenlő, így érdemes egy kis előzetes mintával meghatározni az összes komponenst, majd csak a nagy értékűekre elvégezni a tényleges szimulációt. Meg kell viszont említeni, hogy a gradiens ilyen módon való kiszámítása viszonylag sokáig tart, és kevéssé pontos eredményt ad, ezért a leírt eljárást nem érdemes használni.

6. fejezet

Szukcesszív regressziós approximációk a sztochasztikus programozásban

Az előző fejezetben közölt egydimenziós szukcesszív regressziós approximációk algoritmusát általánosítjuk az n-dimenziós esetre, és megmutatjuk, hogyan lehet ezt alkalmazni a sztochasztikus programozás különböző feladataira. Az előző két fejezetben leírtak az ebben a fejezetben található anyag előkészítéseként tekinthető.

6.1. Sztochasztikus kvadratikus programozás

Megfogalmazunk egy sztochasztikus kvadratikus programozási feladatot, amelyet az SRA algoritmus segítségével meg lehet oldani. Tekintsünk egy általános nemlineáris optimalizálási feladatot:

$$\begin{aligned} &\min f_0(\mathbf{x}) \\ &\text{f.h. } g_1(\mathbf{x}) \leq 0, \\ &\qquad\vdots \\ &\qquad g_m(\mathbf{x}) \leq 0. \end{aligned} \tag{6.1}$$

Ez a feladat jól definiált, ha az $f_0(\mathbf{x}), g_1(\mathbf{x}), i = 1, \ldots, m$ függvények kvázikonvexek. A célfüggvényt teljes általánosságban definiáljuk úgy, hogy az két tagból áll; tartalmaz egy kétlépcsős feladat második lépcsőjéből származó várható pótlás (expected recourse) függvényt és egy további determinisztikus részt:

$$\begin{aligned} f_0(\mathbf{x}) &= q(\mathbf{x}) + F(\mathbf{x}), \\ \text{ahol } q(\mathbf{x}) = E[\mathcal{Q}(\mathbf{x}, \boldsymbol{\xi}_0)] &= E[\min_{\mathbf{y}} \mathbf{q}'\mathbf{y}(\mathbf{x}, \boldsymbol{\xi}_0) \mid T\mathbf{x} + W\mathbf{y}(\mathbf{x}, \boldsymbol{\xi}_0) = \boldsymbol{\xi}_0, \mathbf{y}(\mathbf{x}, \boldsymbol{\xi}_0) \geq 0]. \end{aligned} \tag{6.2}$$

Itt külön feltüntettük, hogy a $q(\mathbf{x})$ várható pótlás függvényének definíciójában szereplő lineáris programozási feladat $\mathbf{y}$ megoldása függ a $\mathbf{x}$ változótól és a $\boldsymbol{\xi}_0$ valószínűségi

változótól is (a továbbiakban ettől eltekintünk). Az $F(\mathbf{x})$ tagot egyszerűség kedvéért definiáljuk egy kvadratikus függvénynek:

$$F(\mathbf{x}) = \mathbf{x}'Q_0\mathbf{x} + \mathbf{b}_0'\mathbf{x}.$$

A feltételek között lehetnek valószínűséggel korlátozott egyenlőtlenségek is és determinisztikus kvadratikus függvényekre felírt egyenlőtlenségek:

$$g_i(\mathbf{x}) = P\{h_i(\mathbf{x}, \boldsymbol{\xi}_i) \leq 0\}, i \in I_1,$$
$$g_i(\mathbf{x}) = \mathbf{x}'Q_i\mathbf{x} + \mathbf{b}_i\mathbf{x} + c_i, i \in I_2,$$

ahol I_1, I_2 az $\{1, 2, \ldots, m\}$ indexek egy tetszőleges felosztása. Ezáltal a számunkra érdekes, a továbbiakban a sztochasztikus kvadratikus programozás alapfeladatának nevezett modell legegyszerűbb változata a

$$\begin{array}{rcl} \min E\{\min \mathbf{q}'\mathbf{y}\} & + & \mathbf{x}'Q_0\mathbf{x} + \mathbf{c}_0'\mathbf{x} \\ \text{f.h. } P\left\{h_i(\mathbf{x}, \boldsymbol{\xi}_i) \leq 0\right\} & \geq & p_i, i \in I_1, \\ \mathbf{x}'Q_i\mathbf{x} + \mathbf{b}_i'\mathbf{x} + c_i & \leq & 0, i \in I_2, \\ \mathbf{x} \geq 0 & & \end{array} \tag{6.3}$$

formába írható, ahol a $0 < p_i < 1$ értékek előre megadott megbízhatósági szintek, $0 < p_i < 1, i \in I_1$, a célfüggvényben pedig a várható pótlás függvényét a szokásos alakjában adtuk meg. A $Q_i, i = 0$, vagy $i \in I_2$ mátrixokról feltesszük, hogy pozitív definitek. Az alapfeladatnak a numerikus megoldására dolgoztuk ki az SRA algoritmust. Természetesen a kvadratikus formák helyett más kvázikonvex függvények is adhatók. A várható pótlás függvényét és a valószínűségi feltételeket véletlen (véletlentől függő) függvényeknek nevezzük, míg a többi függvényt determinisztikusnak nevezzük.

A megoldó algoritmus k-adik lépésében megoldandó közelítő feladatban egy-egy kvadratikus függvénnyel helyettesítjük a várható pótlás függvényét és a valószínűségi korlátos feltételeket, és ezért az SRA-ban megoldandó közelítő feladatok (determinisztikus) kvadratikus optimalizálási problémák.

Megmutatjuk, hogy az általánosan használt sztochasztikus lineáris programozási feladatok ennek a kvadratikus alapfeladatnak a speciális esetei. A valószínűséggel korlátozott modell, amelynek a célfüggvényében csak determinisztikus tagok szerepelnek (a várható pótlás függvénye nem), a következő alakú:

$$\begin{array}{rcl} \min \mathbf{x}'Q_0\mathbf{x} + \mathbf{b}_0\mathbf{x} & & \\ \text{f.h. } P\left\{h_i(\mathbf{x}, \boldsymbol{\xi}_i) \geq 0\right\} & \leq & p_i, i \in I_1, \\ \mathbf{x}'Q_i\mathbf{x} + \mathbf{b}_i\mathbf{x} + c_i & \leq & 0, i \in I_2 \\ \mathbf{x} \geq 0. & & \end{array}$$

Ennek egy változata a széleskörűen alkalmazott STABIL modell legegyszerűbb formája, melyben csak lineáris függvények szerepelnek (lásd a 6.2 részt).

A sztochasztikus kvadratikus programozási alapfeladat speciális eseteként jelenik meg a kétlépcsős modell, amikor a célfüggvényt változatlanul hagyjuk, de nincsen valószínűségi korlátos feltétel:

$$\begin{aligned} \min E[\mathcal{Q}(\mathbf{x}, \boldsymbol{\xi}_0)] &+ \mathbf{x}'Q_0\mathbf{x} + \mathbf{b}_0\mathbf{x} \\ \text{f.h. } \mathbf{x}'Q_i\mathbf{x} + \mathbf{b}_i\mathbf{x} + c_i &\leq 0,\ i \in I_2, \\ \mathbf{x} &\geq 0, \end{aligned}$$

ahol a várható pótlás függvényében szereplő $\mathcal{Q}(\mathbf{x}, \boldsymbol{\xi}_0)$ függvényt a szokásos módon egy lineáris programozási feladat optimális értékeként definiáljuk:

$$\mathcal{Q}(\mathbf{x}, \boldsymbol{\xi}_0) = \{\min_{\mathbf{y}} \mathbf{q}'\mathbf{y} \mid T\mathbf{x} + W\mathbf{y} = \boldsymbol{\xi}_0, \mathbf{y} \geq 0\}.$$

Ebből a feladatból származtatható a kvadratikus feltételi függvények lineárissá egyszerűsítésével az általánosan használt kétlépcsős, sztochasztikus lineáris programozási feladat, melynek formája:

$$\begin{aligned} \min \mathbf{c}'\mathbf{x} &+ E[\mathcal{Q}(\mathbf{x}, \boldsymbol{\xi}_0)] \\ \text{f.h. } A\mathbf{x} &\leq \mathbf{b}, \\ \mathbf{x} &\geq 0. \end{aligned}$$

Végül az alapmodellből származtatható Prékopa vegyes modellje (lásd a 6.4 szakaszt); (i) ha a kvadratikus tagokat lineáris függvényekre egyszerűsítjük, (ii) ha feltesszük, hogy a célfüggvényben szereplő $\boldsymbol{\xi}_0$ valószínűségi változó azonos az egyetlen valószínűségi korlátban szereplő $\boldsymbol{\xi}_1$ valószínűségi változóval $\boldsymbol{\xi}_0 = \boldsymbol{\xi}_1 = \boldsymbol{\xi}$, és (iii) ha a valószínűségi korlátot (a második lépcsőből származó) két U, T mátrix segítségével a következőképpen írjuk fel:

$$\begin{aligned} \min \mathbf{c}'\mathbf{x} &+ E[\mathcal{Q}(\mathbf{x}, \boldsymbol{\xi})], \\ \text{f.h. } A\mathbf{x} &\leq \mathbf{b}, \\ Pr\{U'\boldsymbol{\xi} \geq U'T\mathbf{x}\} &\geq p, \\ \mathbf{x} &\geq 0, \end{aligned} \tag{6.4}$$

ahol a $\mathcal{Q}(\mathbf{x}, \boldsymbol{\xi})$ függvényt egy módosított második lépcsős feladat pótló függvényeként definiáljuk:

$$\begin{aligned} \mathcal{Q}(\mathbf{x}, \boldsymbol{\xi}) &= \min_{\mathbf{y}, \mathbf{z}^+, \mathbf{z}^-} \mathbf{q}'\mathbf{y} + \mathbf{d}^{+'}\mathbf{z}^+ + \mathbf{d}^{-'}\mathbf{z}^- \\ \text{f.h. } W\mathbf{y} + \mathbf{z}^+ - \mathbf{z}^- &= \boldsymbol{\xi} - T\mathbf{x}, \\ \mathbf{y}, \mathbf{z}^+, \mathbf{z}^- &\geq 0. \end{aligned} \tag{6.5}$$

Az így kapott modellek beilleszkednek a Prékopa által megadott általános feladattípusokba (lásd [Pr 95] 233-238 o.).

A kvadratikus alapfeladat két vonatkozásban tér el az általánosan ismert és használt modellektől. Egyrészt mind valószínűségi korlátok, mind a célfüggvénybeli várható pótlás függvénye felléphet ugyanabban a modellben – úgy tűnik, hogy ez (Prékopa vegyes modelljének kivételével) eddig nem szerepelt az irodalomban. Másrészt a gyakorlatban általánosan használt sztochasztikus programozási modellekben a feltételi függvények lineáris függvények voltak; a kvadratikus alapfeladat annyiban tér el ezektől, hogy most explicite megadjuk, hogy kvadratikus függvényekkel ki lehet bővíteni a célfüggvényt és a feltételeket is, még mindig numerikusan megoldható feladatot kapunk.

Ebben a fejezetben először az *SRA* algoritmus három alkalmazási területét írjuk le: a valószínűséggel korlátozott feladatot [De 03c] – 6.2 szakasz, a kétlépcsős feladatot [De 03b], [De 04] – 6.3, 6.6 és 6.7 szakaszok, és Prékopa vegyes feladatát [De 03c] – 6.4 szakasz. Erre az utóbbira eddig nem volt ismeretes megoldó algoritmus. Míg az első két feladattípust használják a sztochasztikus programozási alkalmazások túlnyomó részében, a vegyes modell a kétlépcsős modell egy hiányosságára ad matematikai szempontból használható kezelési módot – a második lépcső feladatának megoldhatóságát csak egy előírt valószínűséggel követeljük meg.

A 6.5 szakaszban közlünk egy Monte Carlo eljárást is, amely segítségével a várható pótlás függvényének értékét lehet hatékonyan kiszámítani többdimenziós (korrelált) normális eloszlás esetén [De 03b]. A 6.6 szakaszban egy közepes méretű kétlépcsős feladat numerikus megoldása során fellépő numerikus nehézségekre adunk eljárásokat. A 6.7 szakaszban pedig véletlenszerűen előállított numerikus feladatok megoldásával megmutatjuk, hogy kétlépcsős feladatok megoldására használható az eljárás, még 100 első lépcsős döntési változó és 120 második lépcsős véletlen változó esetén is [De 04] (ezen feladatok közül egyet megadunk a Függelékben).

Mind a kétlépcsős, mind a valószínűséggel korlátozott feladatok megoldására léteznek megoldó módszerek, de az egyik esetben használt eljárás nem használható a másik típusra. Az általunk javasolt *SRA* eljárás mindkettőre használható. A számítógépes eredmények alapján megfogalmazható az a sejtés, amely az *SRA* eredményének pontosságára vonatkozik: ha az algoritmusban M-szer számítottuk ki a zajos függvényértéket, akkor az eredmény szórásának határértéke $O(1/\sqrt{M})$ nagyságú – és ez a legjobb, amit elvárhatunk.

Érdemes megemlíteni, hogy a kétlépcsős modellek esetén megadott algoritmusok szinte kivétel nélkül csak diszkrét eloszlású valószínűségi változók esetén használhatók (vagy diszkretizálják a folytonos eloszlásokat), – így a magasabb dimenzióban való használhatóságuk elég korlátozott. Az *SRA* algoritmus alkalmazható abban az esetben is, amikor a második lépcsőben diszkrét vagy folytonos eloszlású valószínűségi változók vannak.

Mindegyik algoritmusban az egydimenziós esethez hasonlóan járunk el: 1. a nehezen kiszámítható függvényt egy (kvadratikus) regresszióval helyettesítjük, 2. a becsült függvény segítségével egy közelítő feladatot állítunk elő és 3. a közelítő feladat megoldását hozzáadjuk a regresszió kiszámításában felhasznált ponthalmazhoz. A közölt

numerikus eredmények alapján az SRA algoritmus egy hatékony optimalizálási eljárás a tárgyalt esetekben. Végül megemlítjük, hogy az SRA algoritmussal meg lehet oldani a sztochasztikus kvadratikus programozás fentebb leírt alapfeladatát is. Hátránya az algoritmusnak, hogy konvergenciája még nem bizonyított.

6.2. A STABIL modell numerikus megoldása

6.2.1. Valószínűségi korlátok

A STABIL modellt Prékopa vezette be [PGDP 76], ez a sztochasztikus programozás valószínűséggel korlátozott modelljei közül elsőként alkalmazott korrelált komponensekkel rendelkező normális eloszlást. Ennek a feladattípusnak egy egyszerűsített formája a következő:

$$\begin{aligned} & \min \mathbf{c}'\mathbf{x} \\ G(\mathbf{x}) = P\{\mathbf{t}_i'\mathbf{x} \geq \xi_i, i = 1, \ldots, M\} & \geq p, \\ \mathbf{A}\mathbf{x} & \geq \mathbf{b}, \\ \mathbf{x} & \geq 0, \end{aligned} \tag{6.6}$$

ahol a $\boldsymbol{\xi} = (\xi_1, \ldots, \xi_M)$ valószínűségi változókról feltesszük, hogy együttes eloszlásuk többdimenziós normális eloszlás. Így az eloszlás logkonkáv, tehát kvázikonkáv is, a $G(\mathbf{x}) \geq p$ feltétel megengedett megoldásainak tartománya konvex (lásd Prékopa logkonkávitásra vonatkozó eredményeit [Pr 95]). Az alábbiakban leírt meggondolások és a megoldó algoritmus másmilyen valószínűségi korlátokat tartalmazó feladatokra is alkalmazható, mint például az általános

$$\begin{aligned} & \min f(\mathbf{x}) \\ G(\mathbf{x}) = P\{g_i(\mathbf{x}) \geq \xi_i, i = 1, \ldots, M\} & \geq p, \\ h_i(\mathbf{x}) & \geq 0, i = M+1, \ldots, M+m \end{aligned} \tag{6.7}$$

feladat esetén is (ahol a szereplő g_i, h_i függvények kvázikonkávok, $f(\mathbf{x})$ konvex). Az eljárás nehézség nélkül alkalmazható több valószínűségi korlátot tartalmazó feladatra is. A leírás egyszerűségét szem előtt tartva mi csak az elsőként megfogalmazott feladattal foglalkozunk.

6.2.2. SRA eljárás a STABIL modellre

Tekintsük az előbbi egyszerű (6.6) alakú valószínűségi korlátos feladatot. Az ennek megoldására javasolt algoritmus a következő meggondolásokon alapszik. A nemlineáris $G(\mathbf{x}) \geq p$ valószínűségi feltételt nehéz kiszámítani, és a kiszámításra általában egy Monte

Carlo eljárást használnak, amely egy zajos függvényértéket tud csak előállítani. Ezért a $G(\mathbf{x})$ értékeit meghatározzuk néhány pontban, ezekben a pontokban zajos függvényértékeket számítunk, majd egy regressziós közelítést határozunk meg az előző szakaszban leírtak szerint. A (6.6) feladatban szereplő $G(\mathbf{x})$ függvényt egy regressziós közelítéssel helyettesítve egy közelítő feladatot kapunk, amelyet megoldva a kapott optimális megoldást hozzáadjuk ahhoz a ponthalmazhoz, amely segítségével kiszámítottuk a regressziós függvényt és egy új regressziós közelítést számítunk ki az így kibővített halmazon, s az egész iterációt megismételjük.

A valószínűségi feltételnek a $G(\mathbf{x}) \geq p$ alakú felső nívóhalmazai konvexek, ezért a közelítéshez egy a (6.8) formulában megadott alakú konkáv függvényt fogunk használni.

Az SRA algoritmus formális leírásához vezessük be a következő jelöléseket. Legyen adott az $S_k = \{\mathbf{x}_i, p_i\}_{i=0}^{k-1}$ halmaz, amely valamilyen $\mathbf{x}_i$ pontokat és az ezekben a pontokban kiszámított zajos p_i függvényértékeket tartalmazza, vagyis $Ep_i = G(\mathbf{x}_i), D^2(p_i) = \sigma^2(\mathbf{x}_i)$ – ezt a torzítatlan becslés és a függvényérték közti összefüggést a továbbiakban a $p_i \sim G(\mathbf{x}_i)$ relációval írjuk le röviden.

Bár valójában a p_i becsléseink szórásnégyzete függ attól az $\mathbf{x}$ ponttól, amelyben a függvényérték közelítő kiszámítását végezzük, de az egyszerűség kedvéért az $\mathbf{x}$ argumentumot elhagyjuk és feltesszük, hogy a szórás nem változik, tehát $\sigma^2(\mathbf{x}_i) = \sigma^2$ egy állandó $\sigma > 0$ értékkel. Két okból tehetjük ezt meg. Egyrészt a Monte Carlo számításokban egy kis mintaszámmal közelítőleg megállapíthatjuk az adott pontban a függvényérték kiszámításának a szórását és ennek megfelelően a kívánt (állandó) szórás eléréséhez szükséges mintaszámot kiszámíthatjuk. Másrészt algoritmusainkban olyan pontok kellenek és általában olyan pontokat állítanak elő, amelyek az optimalizálási feladat optimális megoldásának egy kis környezetében vannak, ahol a függvényértékek kiszámításának szórása csak kevéssé változik.

Összefoglalva feltesszük, hogy a függvényértékek kiszámításában egy additív zaj van, $p_i = G(\mathbf{x}_i) + \varepsilon_i$, ahol $E(\varepsilon_i) = 0, D^2(\varepsilon_i) = \sigma^2$, az ε_i valószínűségi változók teljesen függetlenek, így páronként is: $E(p_i p_j) = 0, \forall i \neq j$. Természetesen ezeknek a feltételeknek megfelel az az eset, amikor a $G(\mathbf{x})$ valószínűségi feltétel értékeit egy torzítatlan becslést adó Monte Carlo módszerrel állítjuk elő.

Ha az $S_k = \{\mathbf{x}_i, p_i\}_{i=0}^{k-1}$ halmaz a rendelkezésünkre áll, akkor a

$$q_k(\mathbf{x}) = -\mathbf{x}' D_k \mathbf{x} + \mathbf{b}_k' \mathbf{x} + c_k, \tag{6.8}$$

alakú és L_2-ben minimális normájú közelítést a

$$\min_{D_k, \mathbf{b}_k, c_k} \sum_{i=0}^{k-1} [p_i - q_k(\mathbf{x}_i)]^2. \tag{6.9}$$

optimalizálási feladat megoldásával lehet megadni a következő 6.2.3 szakaszban leírt módon. Ennyi előkészítés után meg tudjuk adni a szukcesszív regressziós approximációk

módszerének a valószínűségi korlátot tartalmazó feladatra adaptált változatát.

Az SRA algoritmus - valószínűségi korlátos feladatra

0. [Előkészítés.] Legyen adott az $S_k = \{\mathbf{x}_i, p_i\}_{i=0}^{k-1}$ halmaz és legyen a k iterációs számláló értéke az adott pontok száma.
1. Számítsuk ki a $q_k(\mathbf{x})$ regressziós függvény $D_k, \mathbf{b}_k, c_k$ együtthatóit az S_k-ból.
2. Helyettesítsük az eredeti (6.6) feladatot a

$$\begin{aligned} &\min \mathbf{c}'\mathbf{x} \\ q_k(\mathbf{x}) = -\mathbf{x}'D_k\mathbf{x} + \mathbf{b}_k'\mathbf{x} + c_k \geq p, \\ &\mathbf{A}\mathbf{x} \geq \mathbf{b}, \\ &\mathbf{x} \geq 0 \end{aligned} \tag{6.10}$$

közelítő feladattal és jelöljük ennek egy optimális megoldását $\mathbf{x}_k$-val.

3. Ha $\mathbf{x}_k$ „elég jó”, akkor STOP. Egyébként számítsuk ki a $p_k \sim G(\mathbf{x}_k)$ zajos függvényértéket, az új $\mathbf{x}_k$ pontot és a p_k értéket csatoljuk az S_k halmazhoz: legyen $S_{k+1} = S_k \cup \{\mathbf{x}_k, p_k\}$, növeljük meg az iterációs számlálót $k := k + 1$ és menjünk vissza az 1. lépésre.

6.2.3. Kvadratikus regresszió n-dimenziós függvények közelítésére

Az ebben a részben alkalmazott jelölések eltérnek az általában használtaktól, mert itt csak azzal az általános kérdéssel foglalkozunk, hogyan lehet egy zajos függvény esetén egy kvadratikus regressziót meghatározni. Ezt az eljárást alkalmazzuk a STABIL feladat, a kétlépcsős feladat és a vegyes modell esetén is.

Tekintsünk egy $f(\mathbf{x}), f : R^n \to R^1$ konvex függvényt, amelynek nem tudjuk a pontos értékét kiszámítani, de a rendelkezésünkre áll egy eljárás, amelynek segítségével tetszőleges $\mathbf{x}_i, i = 1, \ldots, N$, pontban ki tudjuk számítani az $f(\mathbf{x}_i)$ függvényérték egy zajos $p_i, i = 1, \cdots, N$ becslését. A fellépő hibára, illetőleg a függvényérték becslésére pedig feltesszük, hogy a becslés torzítatlan és szórása állandó, vagyis igaz a következő két egyenlőség:

$$E(p_i) = f(\mathbf{x}_i), D^2(p_i) = \sigma^2, \ i = 1, \cdots, N.$$

Az $f(\mathbf{x})$ függvényt egy L_2 minimális normájú kvadratikus regresszióval közelítjük, vagyis feltesszük, hogy a közelítő függvényt

$$q(\mathbf{x}) = \mathbf{x}'Q\mathbf{x} + \mathbf{b}'\mathbf{x} + c = \sum_{r=1}^{n}\sum_{s=1}^{n} q_{rs}x_{ir}x_{is} + \sum_{t=1}^{n} b_t x_{it} + c \tag{6.11}$$

alakban keressük, ahol a Q legyen egy szimmetrikus $n \times n$-es méretű pozitív definit mátrix. A $q(\mathbf{x})$ függvény akkor közelíti L_2 normában a legjobban az $f(x)$ függvényt az $S_N =$

$\{\mathbf{x}_i, p_i\}_{i=1}^N$ pontok és függvényértékek esetén, ha a

$$\min_{q_{\alpha\beta}, b_\gamma, c} \sum_{i=1}^{N} \left[p_i - q(\mathbf{x}_i)\right]^2 \tag{6.12}$$

minimalizálási feladat optimális megoldása. A közelítő $q(\mathbf{x})$ függvény ismeretlen $c, b_\gamma, q_{\alpha\beta}$ paramétereit akarjuk meghatározni, ahol az indexek $\alpha, \beta, \gamma = 1, \ldots, n, \alpha \leq \beta$, a $q_{\alpha\beta} = q_{\beta\alpha}$ a Q mátrix (α, β)-adik eleme, b_γ pedig a $\mathbf{b}$ vektor γ-adik komponense.

Felírjuk a (6.12) feladat esetére az optimalitás elsőrendű szükséges feltételeit – a feladatban szereplő függvényt a $c, b_\gamma, q_{\alpha\beta}$ szerint deriváljuk és egyenlővé tesszük zérussal, ezzel összesen $(n+1)\times(n+2)/2$ lineáris egyenletet kapunk, amely ugyanennyi ismeretlent tartalmaz. Természetesen a (6.12) alakú közelítés paramétereinek meghatározásához legalább $k_{in} = 1 + n + n(n+1)/2$ számú pontra és függvényértékre van szükség, tehát az $N \geq k_{in}$ egyenlőtlenségnek teljesülnie kell. Jelölje $x_{i\alpha}$ az $\mathbf{x}_i$ vektor α-adik komponensét, akkor a lineáris egyenletrendszer a következő:

$$\begin{aligned}
\sum_{i=1}^{n} \left[p_i + \mathbf{x}_i' Q \mathbf{x}_i - \mathbf{b}'\mathbf{x}_i - c\right] &= 0. \\
\sum_{i=1}^{n} \left[p_i + \mathbf{x}_i Q \mathbf{x}_i - \mathbf{b}'\mathbf{x}_i - c\right] x_{i\alpha} &= 0,\ \alpha = 1, \ldots, m, \\
\sum_{i=1}^{n} \left[p_i + \mathbf{x}_i' Q \mathbf{x}_i - \mathbf{b}'\mathbf{x}_i - c\right] x_{i\alpha} x_{i\beta} &= 0,\ \alpha, \beta = 1, \ldots, m, \alpha \leq \beta,
\end{aligned} \tag{6.13}$$

A feladat, illetőleg ennek megoldása átrendezés után egy $\mathbf{M}$ mátrix segítségével is felírható:

$$\mathbf{M}\mathbf{y} = \mathbf{m}, \quad \mathbf{y} = \mathbf{M}^{-1}\mathbf{m} \tag{6.14}$$

alakban, ahol a $\mathbf{y}$ vektor tartalmazza a meghatározandó $c, b_\gamma, q_{\alpha\beta}$ paraméterértékeket, vagyis

$$\mathbf{y}' = (c, b_1, \cdots, b_n, q_{11}, q_{12}, \cdots, q_{1n}, q_{22}, q_{23}, \cdots, q_{2n}, q_{33}, \cdots, q_{nn})$$

Vezessük be a következő jelöléseket. Az $\mathbf{m}$ elemeit a

$$m_0 = \frac{1}{k}\sum_{i=0}^{k-1} p_i,\ m_{1,j} = \frac{1}{k}\sum_{i=0}^{k-1} p_i x_{ij}, m_{2,jl} = \frac{1}{k}\sum_{i=0}^{k-1} p_i x_{ij} x_{il}$$

jelölések segítségével a következő módon lehet részletesen felírni:

$$\mathbf{m}' = (m_0, m_{1,1}, m_{1,2}, \cdots, m_{1,n}, m_{2,11}, m_{2,12}, \cdots, m_{2,1n}, m_{2,22} m_{2,23}, \cdots, m_{2,nn}).$$

Az $\mathbf{M}$ mátrix elemeit az

$$M_{1,j} = \frac{1}{k}\sum_{i=0}^{k-1} x_{ij},\ M_{2,jl} = \frac{1}{k}\sum_{i=0}^{k-1} x_{ij}x_{il},$$
$$M_{3,jlr} = \frac{1}{k}\sum_{i=0}^{k-1} x_{ij}x_{il}x_{ilr},\ M_{4,jlrs} = \frac{1}{k}\sum_{i=0}^{k-1} x_{ij}x_{il}x_{ir}x_{is}$$

jelölések segítségével adhatjuk meg (a fődiagonálisban található első egydimenziós diagonális mátrix tartalma 1, a második, $n \times n$ méretű fődiagonálisban lévő mátrix a másodrendű momentumokat tartalmazza, míg a fődiagonálisban lévő harmadik négyzetes mátrix $n(n+1)/2 \times n(n+1)/2$ méretű és a negyedrendű momentumokat tartalmazza). Az $\mathbf{M}$ mátrix többi elemeit az $M_{1,j}, M_{2,jl}, M_{3,jlr}$ momentumok adják meg a mellékeltek szerint.

A közelítő polinom meghatározásának az $\mathbf{My} = \mathbf{m}$ formából közvetlen invertálással történő meghatározását a direkt módszernek nevezzük – ezt csak a kisebb méretű feladatok megoldásánál használtuk a 6.2 – 6.6 szakaszok numerikus feladataiban. Nagyméretű feladatok megoldásánál (a 6.7 szakaszban) egy kissé más utat követtünk, ezt írjuk le az alábbiakban.

Rövidebb és áttekinthetőbb formában az $\mathbf{M}$ mátrix diádok alkalmazásával írható fel. Legyen adva egy $\mathbf{x} = (x_1, \ldots, x_n) \in R^n$ vektor, amelyből az $n_Q = n(n+1)/2 + n + 1$ dimenziós $\boldsymbol{\xi} = \boldsymbol{\xi}(\mathbf{x})$ vektort a következőképpen alakítjuk ki:

$$\boldsymbol{\xi}' = (1, x_1, \ldots, x_n, x_1^2, 2x_1x_2, \cdots, 2x_1x_n, x_2^2, 2x_2x_3, \cdots, 2x_2x_n, x_3^2, \cdots, 2x_3x_n, \cdots, x_n^2).$$

Ha most az $\mathbf{x}_i, i = 1, \ldots, N$ vektorokból a $\boldsymbol{\xi}_i, i = 1, \ldots, N$ vektorokat alakítjuk ki a fentihez hasonlóan, akkor az

$$\mathbf{M} = \frac{1}{N}\sum_{i=1}^{N} \boldsymbol{\xi}_i\boldsymbol{\xi}_i'$$

összeg adja meg mátrixunkat. Ezek szerint az $\mathbf{M}^{-1}$ inverz létezésének elégséges feltétele, hogy $N \geq k_{in}$ független $\boldsymbol{\xi}_i$ vektort használjunk az $\mathbf{M}$ mátrix előállításában.

$$
\begin{array}{c|ccccccccccccccc}
1 & M_{1,1} & \cdots & M_{1,n} & M_{2,11} & 2M_{2,12} & \cdots & 2M_{2,1n} & M_{2,22} & 2M_{2,23} & \cdots & 2M_{2,2n} & M_{2,33} & \cdots & M_{2,nn} \\
M_{1,1} & M_{2,11} & \cdots & M_{2,1n} & M_{3,111} & 2M_{3,112} & \cdots & 2M_{3,11n} & M_{3,122} & 2M_{3,123} & \cdots & 2M_{3,12n} & M_{3,133} & \cdots & M_{3,1nn} \\
\vdots & & & & \vdots & & & & \vdots & & & & \vdots & & \vdots \\
M_{1,n} & M_{2,1n} & \cdots & M_{2,nn} & M_{3,11n} & 2M_{3,12n} & \cdots & 2M_{3,1nn} & M_{3,22n} & 2M_{3,23n} & \cdots & 2M_{3,2nn} & M_{3,33n} & \cdots & M_{3,nnn} \\
M_{2,11} & M_{3,111} & \cdots & M_{3,11n} & M_{4,1111} & 2M_{4,1112} & \cdots & 2M_{4,111n} & M_{4,1122} & 2M_{4,1123} & \cdots & 2M_{4,112n} & M_{4,1133} & \cdots & M_{4,11nn} \\
2M_{2,12} & 2M_{3,112} & & 2M_{3,12n} & 2M_{4,1112} & 4M_{4,1122} & \cdots & 4M_{4,112n} & 2M_{4,1222} & 4M_{4,1223} & \cdots & 4M_{4,122n} & 2M_{4,1233} & \cdots & 2M_{4,12nn} \\
\vdots & & & & \vdots & & & & \vdots & & & & \vdots & & \vdots \\
2M_{2,1n} & 2M_{3,11n} & \cdots & 2M_{3,1nn} & 2M_{4,111n} & 4M_{4,112n} & \cdots & 4M_{4,11nn} & 2M_{4,122n} & 4M_{4,123n} & \cdots & 4M_{4,12nn} & 2M_{4,133n} & \cdots & 2M_{4,1nnn} \\
M_{2,22} & 2M_{3,122} & \cdots & M_{3,22n} & M_{4,1122} & 2M_{4,1222} & \cdots & 2M_{4,122n} & M_{4,2222} & 2M_{4,2223} & \cdots & 2M_{4,222n} & M_{4,2233} & \cdots & M_{4,2233} \\
2M_{2,23} & 2M_{3,123} & \cdots & 2M_{3,23n} & 2M_{4,1123} & 4M_{4,1223} & \cdots & 4M_{4,123n} & 2M_{4,2223} & 4M_{4,2233} & \cdots & 4M_{4,223n} & 2M_{4,2333} & \cdots & 2M_{4,23nn} \\
\vdots & & & & \vdots & & & & \vdots & & & & \vdots & & \vdots \\
2M_{2,2n} & 2M_{3,12n} & \cdots & M_{3,2nn} & 2M_{4,112n} & 4M_{4,122n} & \cdots & 4M_{4,12nn} & 2M_{4,222n} & 4M_{4,223n} & \cdots & 4M_{4,22nn} & 2M_{4,233n} & \cdots & 2M_{4,2nnn} \\
M_{2,33} & M_{3,133} & \cdots & M_{3,33n} & M_{4,1133} & 2M_{4,1233} & \cdots & 2M_{4,123nn} & M_{4,2233} & 2M_{4,2333} & \cdots & 2M_{4,233n} & M_{4,3333} & \cdots & M_{4,33nn} \\
\vdots & & & & \vdots & & & & \vdots & & & & \vdots & & \vdots \\
M_{2,nn} & M_{3,1nn} & \cdots & M_{3,nnn} & M_{4,11nn} & 2M_{4,12nn} & \cdots & 2M_{4,1nnn} & M_{4,22nn} & 2M_{4,23nn} & \cdots & 2M_{4,2nnn} & M_{4,33nn} & \cdots & M_{4,nnnn}
\end{array}
$$

Vizsgáljuk meg most a felfrissítés kérdését. Tegyük fel, hogy adott k darab $\mathbf{x}_i$ vektor esetén már meghatároztuk az $\mathbf{M} = \mathbf{M}^{(k)} = \frac{1}{k}\sum_{i=0}^{k-1} \boldsymbol{\xi}_i \boldsymbol{\xi}_i'$ mátrixot és annak $\mathbf{M}^{-1}$ inverzét. A kérdés most az, hogyan lehet a $k+1$ darab vektorból előállított $\mathbf{M}^{(k+1)} = \frac{1}{k+1}\sum_{i=0}^{k} \boldsymbol{\xi}_i \boldsymbol{\xi}_i' = \frac{k}{k+1}\mathbf{M} + \frac{1}{k+1}\boldsymbol{\xi}_{k+1}\boldsymbol{\xi}_{k+1}'$ mátrixot és annak $\left[\mathbf{M}^{(k+1)}\right]^{-1}$ inverzét meghatározni. Tekintettel arra, hogy az új $\mathbf{M}^{(k+1)}$ a régi $\mathbf{M}$ mátrixból egy diadikus szorzat hozzáadásával kapható, ezért az

$$(A + \mathbf{u}\mathbf{v}')^{-1} = A^{-1} - \frac{A^{-1}\mathbf{u}\mathbf{v}'A^{-1}}{1 + \mathbf{v}'A^{-1}\mathbf{u}}$$

Sherman-Morrison formulát lehet használni az $\mathbf{M}^{(k+1)}$ inverzének meghatározására:

$$\left[\mathbf{M}^{(k+1)}\right]^{-1} = \left[\frac{k}{k+1}\mathbf{M} + \frac{1}{k+1}\boldsymbol{\xi}_k{\boldsymbol{\xi}_k}'\right]^{-1} = \frac{k+1}{k}\left\{\mathbf{M}^{-1} - \frac{1}{k}\frac{\mathbf{M}^{-1}\boldsymbol{\xi}_k{\boldsymbol{\xi}_k}'\mathbf{M}^{-1}}{1 + \frac{1}{k}{\boldsymbol{\xi}_k}'\mathbf{M}^{-1}\boldsymbol{\xi}_k}\right\}.$$

Ezek szerint az új $\mathbf{M}^{(k+1)}$ mátrix inverze a régi mátrix inverzének és egy diadikus szorzatnak az összege. Ennek segítségével lényegesen lerövidíthető az egyenletrendszer megoldásához szükséges számítási idő a közvetlen módszerhez képest.

Összefoglalva: a $q(\mathbf{x})$ kvadratikus kifejezés paramétereinek meghatározása számítástechnikai szempontból lényegében csak egy $1 + n + n(n+1)/2$ méretű négyzetes mátrix invertálását kívánja. Az ilyen hatványmátrix mindig invertálható, de ha az $\mathbf{x}_i$ pontok egy rögzített ponthoz konvergálnak, akkor a mátrix determinánsa nullához tart, ami numerikus instabilitást okozhat, bár számításaink során ezt nem tapasztaltuk.

A fentiekhez hasonlóan lehet megkonstruálni egy logaritmikus becslést (amely egy kvadratikus $\overline{q}(\mathbf{x})$ függvénnyel közelíti a $\log f(\mathbf{x})$ függvényt) és egy fordított-logaritmikus becslést (amely egy kvadratikus $\overline{\overline{q}}(\mathbf{x})$ függvénnyel közelíti a $\log(1 - f(\mathbf{x}))$ függvényt, lásd [De 98b]), de ezekkel a becslésekkel a továbbiakban nem foglalkozunk, mivel a kvadratikus forma önmagában megfelelő volt, továbbá a logaritmikus transzformáció torzítja a hibát (lásd a 4.2.2 pontban leírtakat).

6.2.4. Numerikus megfontolások

Egy ilyen (6.8) alakú $q_k(\mathbf{x})$ függvény által megadott $\{\mathbf{x} \mid q_k(\mathbf{x}) \geq p\}$ megengedettségi tartomány konvexitását akkor érhetjük el, ha feltesszük, hogy a kvadratikus függvényben szereplő D_k mátrix szimmetrikus és pozitív definit.

A mátrix szimmetrikussága minden további nélkül biztosítható azzal, hogy csak a fődiagonális és a felette (vagy alatta) lévő elemek kiszámítását végezzük el, majd tükrözzük a kapott értékeket a fődiagonálisra.

A pozitív definitséget nem ellenőrizzük és nem is tudjuk biztosítani az algoritmus folyamán. A gyakorlati számításokban a pozitív definitséget azzal érjük el, hogy az optimumtól „messze” lévő pontokat is beveszünk a kezdeti S_k halmazba, illetőleg hagyjuk az algoritmust, hogy önjavítólag korrigálja a függvényt. Ezeknek a „messze” lévő pontoknak a

tapasztalatok alapján elég volt olyan pontokat használni, amelyek esetén a $|G(\mathbf{x})-p| > 0.1$ volt.

Egy másik – a számítógépes algoritmusban használt – eljárás a „messzi" pontok automatikus meghatározására az lehet, hogy a megoldások minden komponensére előírunk egy elég nagy alsó és felső korlátot $K_L \leq |\mathbf{x}|_i \leq K_U, i = 1, \ldots, n$. Ha a számítások folyamán a D_k mátrix nem pozitív definit (vagy a megfelelő mátrix invertálása nem sikerül), akkor is tovább folytatjuk az iteráció számításait. A közelítő feladat még mindig adni fog valamilyen „megoldást", amely megengedett megoldás lesz a determinisztikus feltételekre, de valószínűleg nem megengedett a valószínűségi feltételre. Ez a „megoldás" nyilván nem optimális és legalább néhány komponensében egyenlő lesz a K_L vagy K_U korláttal. Ha ezt a „messze" lévő „megoldást" (és az ezen felvett függvényértéket) hozzáadjuk az S_k halmazhoz, akkor ez javítja az ezek után kiszámításra kerülő D_k mátrixok pozitív definitségét – tehát stabilizálja a $q_k(\cdot)$ kvadratikus kifejezést. A gyakorlati számításokban ilyen „megoldások" nagyon ritkán fordulnak elő egy $n \leq 10$ dimenziós közelítésben, mindössze néhányszor, az első 100-300 iterációban, az utána következő 10000 vagy még több iterációban a $q_k(\cdot)$ stabilnak mutatkozott és nem fordult elő, hogy nem pozitív definit volt a mátrix az általunk vizsgált numerikus feladatokban. A nagyobb méretű feladatoknál ez sokkal gyakoribb volt, de a fentebbiekben leírt „megoldások" használata továbbra is eredményes volt.

Az algoritmus 3. lépésében szereplő „elég jó" kifejezés többféleképpen is értelmezhető (az *SRA* algoritmus más feladatokra vonatkozó alkalmazásaiban is fennállnak az alábbiak). Az erre vonatkozó lehetőségek közül csak néhányat említünk. Az egyik megállási kritérium az lehet, hogy az $\mathbf{x}_k$ pontban meghatározzuk a $G(\mathbf{x})$ függvény zajos gradiensét és ha ez egy adott ε tolerancia-határon belül merőleges az aktív feltételek halmazára (az érintő síkjukra), akkor megállhatunk. Használhatók bonyolultabb megállási kritériumok is, például a Shapiro és Homem-de-Mello [SH 98] által megadott megállási szabályok, vagy a Mak, Morton és Wood által megfogalmazott, optimum-közeliségre vonatkozó statisztikai tesztek [MMW 99]. A 6.2, 6.3, 6.4 és 6.6 szakaszban megadott számítógépes futásaink alkalmával egy rögzített iterációszámot adtunk meg, a 6.7 szakasz nagyméretű feladatai esetén egy heurisztikus megállási szabályt alkalmaztunk, amely eredményét a Mak-Morton-Wood cikkben [MMW 99] található konfidenciaintervallum megalkotásával is minősítettük (a részletes leírást lásd ott).

Lehetőség van arra, hogy a megoldások pontosságára vonatkozó (később részletesen kifejtendő) sejtésünk alapján írjunk elő a pontosságra egy korlátot. A kis és közepes méretű feladatainkra vonatkozó számítógépes futásainkban egyszerűen megadtunk egy rögzített egész értéket az iterációk (előállított pontok) számára.

A számításainkban szereplő mennyiségeknek a felfrissítésére a közepes vagy annál kisebb méretű feladatokra a közvetlen eljárást alkalmaztuk a 6.2 – 6.6 szakaszokban. A (6.14) egyenletben szereplő átlagok helyett természetesen a

$$\sum_i \sum_{s_1} x_{is_1}, \cdots, \sum_i \sum_{s_1} \sum_{s_2} \sum_{s_3} \sum_{s_4} x_{is_1} x_{is_2} x_{is_3} x_{is_4}$$
$$\sum_i p_i, \ \sum_i \sum_{s_1} p_i x_{is_1}, \ \sum_i \sum_{s_2} \sum_{s_3} p_i x_{is_2} x_{is_3}$$

értékeket tároltuk, a felfrissítés végrehajtásához ezekhez csak az új $\mathbf{x}_k, p_k$ értékekből számított tagokat kellett hozzáadni és $(k+1)$-gyel mindegyiket elosztani ahhoz, hogy a (6.13) egyenletrendszer együtthatóit megkapjuk, kisméretű példáinkban a sebesség nem volt fontos.

6.2.5. Számítógépes eredmények

Az ebben a fejezetben megadott számítógépes eredményeket (a 6.6 és 6.7 részben leírt futási eredmények kivételével) ugyanazon számítógépes környezetben értük el. Egy személyi számítógépet használtunk, amelynek 133 Mhz-es processzora volt, belső memóriája 64 MB RAM. A közelítő feladatokban fellépő lineáris és kvadratikus optimalizálási feladatokat a standard MINOS programcsomag segítségével oldottuk meg, a feladatok megadásánál a teljes formátumot használtuk, vagyis nem használtuk ki a ritka mátrixokra vonatkozó opciókat. A lineáris programozási feladatoknál a „HOT" opciót használtuk, mivel sokszor majdnem azonos feladatokat kellett megoldani – változó jobboldali vektor esetén.

Tekintsük a következő numerikus példát:

$$\begin{aligned} \min f(x_1, x_2) &= \min x_1 + x_2 \\ P\left\{ \begin{array}{rcrcrcl} 3x_1 & + & x_2 & - & 6 & \geq & \beta_1 \\ x_1 & + & 8x_2 & - & 8 & \geq & \beta_2 \end{array} \right\} &\geq p, \\ x_1 + 4x_2 &\geq 4, \\ 3x_1 + x_2 &\geq 3, \\ x_1 \geq 0, \qquad & x_2 \geq 0, \end{aligned}$$

ahol β_1, β_2 normális eloszlású valószínűségi változók, várható értékük 0, szórásuk 1, korrelációs együtthatójuk ϱ, a példa a [De 72] cikkből származik.

Különböző számítógépes futások eredményeit adjuk meg a 6.1 táblázatban. Az első oszlop az iterációk számát adja, a második a használt módszer, a harmadik a kapott $\mathbf{x}_{STO}$ optimális megoldást, a negyedik az optimális célfüggvényértéket tartalmazza. A $\sigma_P = \sigma_1/\sqrt{k}$ alatti oszlop adja a sejtés alapján a $P\{\cdot\}$ valószínűségi korlát értékének szórását, az utolsó oszlop pedig a valószínűségi korlátban szereplő $P\{\cdot\}$ valószínűség „pontos" értékét tartalmazza – ezt a „pontos" értéket úgy kaptuk, hogy a függvénykiszámítási eljárást az

utolsó pontban megismételtük megnövelt mintaszámmal, vagyis az ebben az oszlopban található valószínűségek becslések, melyek szórása 0.0002 volt.

Az első sorban a Szántai által kifejlesztett sztochasztikus programozási programcsomag által adott eredményt közöljük, a második sor azt az eredményt mutatja, amelyet az SRA algoritmus ért el determinisztikus eloszlásfüggvényértékek esetén (a valószínűségi korlát most egy kétdimenziós normális eloszlásfüggvény, amelynek értékeit egy sorbafejtéssel pontosan meg lehet határozni). A Szántai-féle és az SRA_{det} által adott optimális megoldásokon a célfüggvényértékek különbsége Δ=2.9140-2.8992=0.015; ez az SRA algoritmusnak, mint nemlineáris optimalizálási algoritmusnak az 1000 iteráció utáni hibáját mutatja (determinisztikus függvényértékek esetén). A 6.1 táblázat második részében az SRA által adott eredményeket találjuk különböző k iterációs számra.

A 6.2 táblázatban változó p valószínűségek és korrelációs együtthatók eseteire kapott eredményeket közlünk. Egy zajos függvényérték kiszámításának szórása itt is $\sigma_1 = 0.01$ volt, az iterációk számát $k = 50$-nek vettük. Vegyük észre, hogy az utolsó oszlopban lévő valószínűségek tényleges eltérése a p korláttól nagyon közel van a $3\sigma_P = 0.0045$ értékhez.

Megfogalmazzuk sejtésünket. Az egész eljárás folyamán a valószínűségi korlátban szereplő $P\{\cdot\}$ értékét egy Monte Carlo eljárással számítottuk (az egyszerűség kedvéért a durva módszert használtuk), amely egy függvényérték becslést $\sigma_1 = 0.01$ szórással állított elő. Sejtésünk az, hogy a k iterációs lépés után kapott $\mathbf{x}_k$ pontban a $G(\mathbf{x}_k)$ pontos függvény érték és a p korlát értéke közötti eltérés (vagyis a tényleges hiba) körülbelül $3\sigma_P = 3\sigma_1/\sqrt{k}$ nagyságú. Tekintsük például az utolsó sorban adott eredményeket. Itt a tényleges hiba $|G(\mathbf{x}_k) - p| \leq 3\sigma_P = 0.0012$ nagyságú lehet, ami a táblázatban található 0.0025 eltéréshez igen közel van, annak ellenére, hogy az egy függvénykiszámításban várható hiba $3\sigma_1 = 0.03$ lenne.

Más szóval a numerikus eredmények alapján azt tapasztaltuk, hogy k lépés után majdnem mindig fennáll az $|G(\mathbf{x}_k) - p| \leq 3\sigma_P$ egyenlőtlenség a $\sigma_P = \sigma_1/\sqrt{k}$ jelöléssel. Némileg leegyszerűsítve azt az állítást mondhatjuk ki, hogy a végső eredmény szórása σ_P. Ezt a sejtést az optimális szóráscsökkenés sejtésének nevezzük – ugyanis ez a legjobb, amit statisztikai mintavételezéssel elérhetünk (ha az iteratív eljárásban előállított minden pont ugyanott lenne, az $\mathbf{x}_k$ pontban, akkor kapnánk ezt a szóráscsökkenést, pedig ezek a pontok nem azonosak).

Mint majd látni fogjuk, ez a sejtés fennáll az SRA más alkalmazásainál is, nagyobb iterációszámnál jobban látható ennek a sejtésnek a teljesülése. Jól látható ennek a sejtésnek a helyessége a 6.2 táblázatban közölt eredmények esetében is.

6.3. A kétlépcsős feladat megoldó algoritmusa

6.3.1. A kétlépcsős feladat

A valószínűséggel korlátozott feladatok mellett a sztochasztikus programozás másik legfontosabb feladattípusa a kétlépcsős feladat (recourse problem). A formális leíráshoz a

$k =$	módszer	$\mathbf{x}_{STO}$	$f(\mathbf{x}_{STO})$	σ_P	$P\{\mathbf{x}_{STO}\}$
	Szántai	(1.9977, 0.9015)	2.8992	–	0.8000
1000	SRA_{det}	(1.9686, 0.9455)	2.9140	–	0.8003
20	SRA	(1.9831, 0.9678)	2.9509	0.0030	0.8305
50	SRA	(1.9663, 0.9732)	2.9392	0.0014	0.8019
200	SRA	(1.9543, 0.9809)	2.9353	0.0010	0.7923
1000	SRA	(1.9555, 0.9795)	2.9351	0.0004	0.8025

6.1. táblázat. Valószínűséggel korlátozott feladat: Szántai és az SRA eredményei $p = 0.8, \varrho = 0.9$ esetén, egy függvénykiszámítás szórása $\sigma_1 = 0.01$.

$p =$	$\varrho =$	$\mathbf{x}_{STO}$	$f(\mathbf{x}_{STO})$	$P\{\mathbf{x}_{STO}\}$
0.80	-0.9	(2.0087, 0.9708)	2.9794	0.8003
0.80	-0.2	(2.0075, 0.9697)	2.9772	0.8098
0.80	0.0	(2.0104, 0.9629)	2.9733	0.8080
0.80	0.2	(2.0179, 0.9496)	2.9675	0.8048
0.80	0.5	(2.0398, 0.9163)	2.9562	0.7962
0.80	0.9	(2.0820, 0.8581)	2.9402	0.7940
0.95	0.2	(2.2623, 0.9615)	3.2238	0.9432
0.95	0.9	(2.3405, 0.9232)	3.2638	0.9510

6.2. táblázat. Valószínűséggel korlátozott feladat: SRA eredmények különböző p és ϱ értékekre, $k = 50$, $\sigma_1 = 0.01$, $\sigma_P = \sigma_1/\sqrt{k} = 0.0015$.

pótló (recourse) függvényt definiáljuk először

$$\mathcal{Q}(\mathbf{x}, \boldsymbol{\xi}) = \min_{\mathbf{y}} \{\mathbf{q}'\mathbf{y} \mid W\mathbf{y} = \boldsymbol{\xi} - T\mathbf{x}, \mathbf{y} \geq 0\}. \tag{6.15}$$

A $\mathcal{Q}(\mathbf{x}, \boldsymbol{\xi})$ függvény definíciójában szereplő lineáris programozási feladatot a második lépcső feladatának nevezzük, ahol $T, W, \mathbf{q}$ rögzítettek, a $\boldsymbol{\xi}$ valószínűségi változóról feltesszük, hogy eloszlásfüggvénye $\Phi(\cdot)$, a tartóját pedig Ξ jelöli. A várható pótlás (expected recourse) függvényét a

$$Q(\mathbf{x}) = E(\mathcal{Q}(\mathbf{x}, \boldsymbol{\xi})) = \int_{\Xi} \mathcal{Q}(\mathbf{x}, \boldsymbol{\xi}) d\Phi(\boldsymbol{\xi})$$

egyenlőséggel adjuk meg. Az első lépcső feladata ekkor

$$\begin{array}{rcl} \min\ \mathbf{c}'\mathbf{x} & + & Q(\mathbf{x}), \\ \text{f.h. } A\mathbf{x} & = & \mathbf{b}, \\ \mathbf{x} & \geq & 0, \end{array} \tag{6.16}$$

ahol $A, \mathbf{b}, \mathbf{c}$ rögzítettek. Feltesszük, hogy a kétlépcsős feladat véges optimumának és az optimális megoldásának létezéséhez minden feltétel teljesül, vagyis a teljes pótlás (complete recourse) igaz – tetszőleges $\mathbf{x}$ esetén a második lépcsős feladatnak van véges optimuma és megengedett megoldása (vagy legalábbis relative teljes a pótlás). Az A, T, W mátrixok $m_1 \times n_1, m_2 \times n_1, m_2 \times n_2$ méretűek, a $\mathbf{b}$ és $\boldsymbol{\xi}$ vektorok m_1 illetőleg m_2 dimenziósok. Az erre a feladatra vonatkozó gazdag irodalomból csak a [KW 94], [May 98], [Pr 95] könyveket említjük.

A kétlépcsős feladatok numerikus megoldásában a legnagyobb nehézséget a várható pótlás definíciójában szereplő várható érték kiszámítása jelenti, mivel ez egy többdimenziós integrál a $\boldsymbol{\xi}$ valószínűségi változó Ξ tartója felett. Elméletileg ez pontosan kiszámítható diszkrét tartó esetén, de folytonos eloszlású valószínűségi változó esetén a kiszámítás nem könnyű feladat. A fellépő nehézségek megoldására vagy diszkretizálási eljárásokat használnak, vagy pedig a numerikus számításokban használható egyszerű alsó és felső korlátokat adnak meg a pótlás függvényére (implicite ezekben az eljárásokban is diszkretizálást használnak). Diszkretizálás esetén hatalmas méretű lineáris programozási feladat keletkezhet, amelynek a megoldása külön eljárások kifejlesztését igényelte, igényli. Itt a Dantzig-Wolfe dekompozíciót, a Wets-van Slyke javasolta L-alakú eljárást (a Benders dekompozíciót), a Ruszczyński-féle regularizált dekompozíciót és a Higle és Sen által javasolt véletlen lineáris függvényeket alkalmazó sztochasztikus dekompozíciót kell megemlíteni.

Ezek a megközelítések általában kevéssé hatékonynak mutatkoznak a folytonos esetekben, sőt még nagyszámú pontból álló diszkrét tartó esetén is nehezen megoldhatók a keletkezett feladatok (bár itt többféle eljárást alkalmaznak a numerikus hatékonyság megőrzésére, mint például a véletlen vektorok csoportosítása, vagy kiszűrése). A normális eloszlású valószínűségi vektorváltozó használata a (6.15) alatti lineáris programozási feladatban pontosan az említett nehézségeket mutatja, különösen akkor, ha (erősen) korrelált

komponensek szerepelnek az eloszlásban. Gyakorlatilag ugyanolyan nehézzé válik a feladat megoldása, mint a többdimenziós halmazok valószínűségének kiszámítása.

6.3.2. Az SRA algoritmus a kétlépcsős feladatra

A várható pótlás függvényében szereplő várható értéket nehéz kiszámítani. De egy tetszőleges $\mathbf{x}$ pont és a $\boldsymbol{\xi}$ valószínűségi változó tetszőleges $\boldsymbol{\xi}_i$ realizációja esetén ki lehet számítani a q_i függvényértéket, amely a következő feladat optimális célfüggvényértéke:

$$\begin{aligned} q_i = \mathcal{Q}(\mathbf{x}, \boldsymbol{\xi}_i) &= \min_{\mathbf{y}} \mathbf{q}'\mathbf{y} \\ \text{f.h. } T\mathbf{x} + W\mathbf{y} &= \boldsymbol{\xi}_i, \\ \mathbf{y} &\geq 0. \end{aligned}$$

Itt a q_i érték a $Q(\mathbf{x})$ függvény egy zajos becslésének tekinthető, mivel $Eq_i = Q(\mathbf{x})$. A q_i függvényérték $D^2(q_i) = \sigma^2(\mathbf{x})$ szórásnégyzete függ az $\mathbf{x}$ ponttól, de a valószínűséggel korlátozott modell kapcsán leírtakhoz hasonlóan feltehetjük, hogy ez a szórás független az $\mathbf{x}$-től, $D^2(q_i) = \sigma^2$ a továbbiakban.

Ezek szerint a következő Monte Carlo eljárás adható a várható pótlás $Q(\mathbf{x})$ függvényértékének a meghatározására. Legyen $\mathbf{x}_i$ adott és generáljuk a $\boldsymbol{\xi}$ valószínűségi változó $\boldsymbol{\xi}_{ij}, j = 1, \ldots, M$ független realizációit. Határozzuk meg a

$$q_i = \frac{1}{M} \sum_{j=1}^{M} \mathcal{Q}(\mathbf{x}_i, \boldsymbol{\xi}_{ij})$$

értéket, amely $Q(\mathbf{x}_i)$ függvényérték egy torzítatlan becslése. Ez az eljárás lehetőséget ad arra, hogy egy adott $\{\mathbf{x}_i\}_{i=0}^{k-1}$ ponthalmaz esetén kiszámítsuk az ezekhez a pontokhoz tartozó zajos q_i függvényértékeket. Így feltehetjük, hogy az SRA algoritmus megkezdése előtt a rendelkezésünkre áll az $S_k = \{\mathbf{x}_i, q_i\}_{i=0}^{k-1}$ pont-függvényérték halmaz. A nehezen meghatározható $Q(\mathbf{x})$ függvényt azon

$$q_k(\mathbf{x}) = \mathbf{x}' D_k \mathbf{x} + \mathbf{b}_k' \mathbf{x} + c_k \tag{6.17}$$

alakú kvadratikus függvénnyel közelítjük, amely a

$$\min_{D_k, \mathbf{b}_k, c_k} \sum_{i=0}^{k-1} [q_i - q_k(\mathbf{x}_i)]^2. \tag{6.18}$$

optimalizálási feladat megoldása Az SRA algoritmus formális leírása a következő:

`Az` SRA `algoritmus - kétlépcsős feladatra`

0. `[Előkészítés.] Legyen adott az` $S_k = \{\mathbf{x}_i, q_i\}_{i=0}^{k-1}$ `halmaz és legyen a` k `iterációs számláló az adott pontok száma.`

1. Számítsuk ki a $q_k(x)$ függvény $D_k, \mathbf{b}_k, c_k$ együtthatóit az S_k-ból.
2. Helyettesítsük az eredeti (6.16) feladatot a

$$\begin{aligned} \min_{\mathbf{x}} \mathbf{c}'\mathbf{x} + q_k(\mathbf{x}), \\ A\mathbf{x} \leq \mathbf{b}, \\ \mathbf{x} \geq 0, \end{aligned} \tag{6.19}$$

közelítő feladattal és jelöljük egy optimális megoldását $\mathbf{x}_k$-val.
3. Ha $\mathbf{x}_k$ „elég jó'', akkor STOP. Egyébként számítsuk ki a $q_k \sim Q(\mathbf{x}_k)$ zajos függvényértéket, az új $\mathbf{x}_k$ pontot és a q_k értéket csatoljuk az S_k halmazhoz: legyen $S_{k+1} = S_k \cup \{\mathbf{x}_k, q_k\}$, növeljük meg az iterációs számlálót $k := k + 1$ és menjünk vissza az 1. lépésre.

A megállási szabály, a D_k mátrix pozitív definitsége, a legkisebb négyzetek módszerének alkalmazhatósága, a felfrissítési szabályok tekintetében teljesen ugyanúgy járunk el, mint azt a valószínűségi korlátos modellek esetén már leírtuk.

6.3.3. Számítógépes eredmények

Az itt közölt numerikus példát [De 02] Mayer állította elő, az általa és Kall által kifejlesztett SLP-IOR [May 92], [May 98] programcsomag segítségével. A feladatot Mayer diszkretizálta: 319 és 522 lehetséges értéket vehettek fel a valószínűségi változók. Az így diszkretizált feladatokon elért eredményeket a következő táblázat első két sora tartalmazza. A feladatot kis mérete ellenére viszonylag nehéz megoldani, amit az is mutat, hogy egy véletlen realizáció esetén a várható pótlás becslésének $D^2[\mathcal{Q}((\mathbf{x}_k, \boldsymbol{\xi})] \sim 15^2$ a szórásnégyzete, ha $\mathbf{x}_k$ közel van az optimális megoldáshoz. (A 6.7 szakaszban bevezetett nehézségi fok ezen példa esetén $3D[\mathcal{Q}((\mathbf{x}_k, \boldsymbol{\xi})]/f(\mathbf{x}_k) = 1.7$ volt – a függvényérték kiszámításának hibája a függvény értékével egyező nagyságrendű volt.) A számítógépes futások egyéb részleteit és további numerikus példákat az *SRA* működésére a [De 02] cikkben lehet megtalálni, köztük egy véletlen elemekkel rendelkező T mátrixot tartalmazó feladatot is. A numerikus példa a következő:

$$\begin{array}{lrrrrrcl} & \min\,(9.0x_1 & +8.1x_2 & +E(3.6y_1 & +7.4y_2 & +6.9y_3)) & & \\ \text{f.h.} & 2.5x_1 & +1.6x_2 & & & & \geq & 1.8 \\ & 9.4x_1 & +9.0x_2 & & & & \geq & 8.0 \\ & 6.0x_1 & -9.2x_2 & -0.9y_1 & -0.7y_2 & +1.7y_3 & = & \xi_1 \\ & -6.3x_1 & -1.2x_2 & +3.9y_1 & +9.0y_2 & -13.0y_3 & = & \xi_2 \\ & & x_1, x_2, & y_1, y_2, y_3 & \geq & 0 & & \end{array}$$

	$\mathbf{x}$	$\mathbf{c}'\mathbf{x}+E(\mathbf{q}'\mathbf{y})$	"pontos"	$M=$	$k=$
$discr_{319}$	(0.9655, 0.0)	26.9805			
$discr_{522}$	(0.9621, 0.0)	26.9847			
SRA_1	(1.0286,0.0)	28.14($\pm$1.41)	27.39($\pm$0.57)	100	100
SRA_2	(0.9199,0.0)	26.94($\pm$0.68)	27.27($\pm$0.06)	400	100
SRA_3	(0.9663,0.0)	26.44($\pm$0.53)	27.01($\pm$0.13)	500	500
SRA_4	(0.9892,0.0)	26.92($\pm$0.43)	26.95($\pm$0.14)	900	400
SRA_5	(0.9291,0.0)	27.60($\pm$0.32)	27.14($\pm$0.06)	400	1000
SRA_6	(0.9412,0.0)	27.44($\pm$1.35)	26.93($\pm$0.14)	100	8000

6.3. táblázat. Két-lépcsős feladat: Mayer eredményei és az SRA eredményei különböző mintaszám és iterációs számok esetén.

Az első két egyenlőtlenség tartalmazza az első lépcsős feladatot, a harmadik és a negyedik egyenlőtlenség pedig a második lépcső feladatát adja meg. Vegyük észre, hogy a feladat teljes pótlású. A véletlen jobboldali (ξ_1, ξ_2) valószínűségi változók együttes eloszlása normális, $(\xi_1, \xi_2) \in N((5.8, -8.7), \Sigma)$, ahol $\Sigma_{1,1} = D^2(\xi_1) = 1, \Sigma_{2,2} = D^2(\xi_2) = 1, \Sigma_{1,2} = Corr(\xi_1\xi_2) = 0.9$.

A 6.3 táblázatban adjuk meg a Mayer által két különböző diszkretizálás esetén kapott eredményeket és az SRA algoritmus által, különböző iterációs számokra, illetőleg különböző mintaszámra kapott eredményeket. Az SRA algoritmusban a várható pótlás értékét egy M mintaszámot használó durva Monte Carlo módszer segítségével számítottuk ki.

A táblázat első oszlopában a használt módszert tüntettük fel, az $\mathbf{x}$ oszlopban a k számú iteráció elvégzése után kapott optimális megoldást adtuk meg, a $\mathbf{c}'\mathbf{x}+E(\mathbf{q}'\mathbf{y})$ feladat alatt az első lépcsős feladat célfüggvényének értéke van az utolsó $\mathbf{x}$ pontban. A „pontos" oszlop tartalmazza a megnövelt mintaszámmal újraszámított célfüggvényértéket (itt 100000 véletlen mintát használtunk), a számadatok mellett álló ($\pm$1.41) alakú kifejezések azt mutatják, hogy az adott függvényérték kiszámolt értékének mekkora a szórása. Egy (várható pótlás) függvényérték kiszámításához M mintát használtunk, és k az iterációk száma.

A 6.4 táblázatban egyetlen számítógépes futás rész-eredményeit közöljük. Itt $\mathbf{x}_k$ a k-adik iterációban kapott közelítő eredmény, $\mathbf{c}'\mathbf{x}_k+E(\mathbf{q}'\mathbf{y})$ az aktuális célfüggvény értéke, valamint a „pontos" (újraszámított) eredmény található, az M mintaszámmal. Végül az $\overline{\mathbf{x}}$ oszlopában a súlypont aktuális értékét adjuk meg (ez lényegesen lassabban konvergál, de nagy ingadozásoktól mentes).

A 6.4 táblázatban megadott eredmények ismételten szemléltetik a valószínűségi korlátos feladat numerikus eredményeinél leírt sejtésünket: a k-adik iteráció után kapott tényleges hiba kisebb mint $3\sigma_Q = 3\sigma_1/\sqrt{k}$, hiszen most $\sigma_1 = 1.41$, $k = 40000, 3\sigma_Q = 3 \cdot 0.007 = 0.02$. A tényleges hibát pedig a következőképpen határozhatjuk meg: jelölje $\mathbf{x}_{opt}$ az optimális megoldást, $f(\mathbf{x}_{opt})$ a tényleges optimumot, $\mathbf{c}'\mathbf{x}_k + E(\mathbf{q}'\mathbf{y})$ a közelítő op-

	$\mathbf{x}$	$\mathbf{c}'\mathbf{x}_k + E(\mathbf{q}'\mathbf{y})$	"pontos"	$\overline{\mathbf{x}}$
$disz_{319}$	(0.9655, 0.0)	26.9805		
$disz_{522}$	(0.9621, 0.0)	26.9847		
$k = 1$	(1.1479, 0.0)	34.79($\pm$1.41)	32.53($\pm$0.14)	(1.056,-0.11)
$k = 10$	(1.0867, 0.0)	29.81($\pm$1.41)	29.67($\pm$0.14)	(1.095,-0.05)
$k = 100$	(1.0439, 0.0)	29.40($\pm$1.41)	28.26($\pm$0.14)	(1.063,-0.01)
$k = 1000$	(1.0188, 0.0)	29.95($\pm$1.41)	27.68($\pm$0.14)	(1.029, 0.00)
$k = 4000$	(1.0042, 0.0)	29.44($\pm$1.41)	27.43($\pm$0.14)	(1.015, 0.00)
$k = 10000$	(0.9842, 0.0)	26.93($\pm$1.41)	27.19($\pm$0.14)	(1.003, 0.00)
$k = 40000$	(0.9615, 0.0)	27.59($\pm$1.41)	27.09($\pm$0.14)	(0.975, 0.00)

6.4. táblázat. Kétlépcsős feladat: az *SRA* algoritmus egy futás k-adik lépésében kapott eredményei, $M = 100, \sigma_1 = 1.41$. A „pontos" függvényérték az utolsó pontban 26.92($\pm$0.02).

timális célfüggvényérték σ_a szórású becslését, akkor az elméleti minimum és az általunk adott közelítésen felvett célfüggvényérték eltérése a

$$|f(\mathbf{x}_{opt}) - [\mathbf{c}'\mathbf{x}_k + E(\mathbf{q}'\mathbf{y})]| \leq 3\sigma_a + 3\sigma_Q = 3 \cdot 0.02 + 0.02 = 0.08$$

egyenlőtlenségnek kell teljesülnie – ami láthatóan fennáll. Jegyezzük meg, hogy ez úgy következett be, hogy csak $\sigma_1 = 1.41$ szórású függvényértékeket tudtunk számolni, tehát a hiba 4.2 nagyságú is lehetne.

A fentebbi numerikus feladat megoldása $M = 100$ mintaszámmal és $k = 100$ iterációs számmal egy 133 MHz-es személyi számítógépen mintegy 30 másodpercig tartott.

A kvadratikus $q_k(\cdot)$ közelítés paraméterei az első 500 iteráció után a következők voltak:

$$D = \begin{pmatrix} 49.12 & -59.86 \\ -59.86 & 144.72 \end{pmatrix}, \ \mathbf{b} = \begin{pmatrix} -100.3 \\ 139.0 \end{pmatrix}, \ c = 70.38$$

Az első lépcső (9.0,8.4) költségvektorát ehhez hozzáadva a közelítő célfüggvény $g_k(\mathbf{x}) = \mathbf{c}'\mathbf{x} + q_k(\mathbf{x})$, amelynek a (nemnegatívitási feltételek figyelmen kívül hagyásával kapott) minimum pontja a $\mathbf{x}_{min} = (0.624, -0.250)$ pontban van, a közelítő célfüggvényérték pedig $g_k(\mathbf{x}_{min}) = 23.5$.

Az algoritmussal nehézség nélkül lehet kezelni azt az esetet is, amikor a második lépcsőben szereplő T mátrixban valószínűségi változók vannak. Megváltoztatjuk az előző numerikus példát úgy, hogy a T mátrix elemeihez nulla várható értékű valószínűségi változókat adunk hozzá. Tegyük fel, hogy u_1, u_2 független, a $[0, 1]$ intervallumban egyenletes eloszlású valószínűségi változók, a randomizált $\widetilde{T}$ mátrix álljon a következő elemekből:

	$\mathbf{x}$	$\mathbf{c}'\mathbf{x} + E(\mathbf{q}'\mathbf{y})$	"pontos"	$M =$	iter.szám
SRA_1	(1.0188,0.0)	29.09($\pm$1.66)	32.07($\pm$0.14)	100	400
SRA_2	(0.9355,0.0)	26.03($\pm$1.33)	28.11($\pm$0.14)	100	400

6.5. táblázat. Kétlépcsős feladat véletlen $\widetilde{T}$ mátrixszal, különböző kiindulási halmazok esetén.

$$
\begin{aligned}
u_1, u_2 &\in \mathcal{U}(0,1), \\
\tilde{t}_{1,1} = t_{1,1} + u_1 + u_2 - 1, &\qquad \tilde{t}_{1,2} = t_{1,2} + u_2 - 0.5 \\
\tilde{t}_{2,1} = t_{2,1} + u_2 - u_1, &\qquad \tilde{t}_{2,2} = t_{2,2} + u_1 - 0.5
\end{aligned}
$$

Eredményeinket a mellékelt 6.5 táblázatban foglaljuk össze.

A következő numerikus példát a Kall és Wallace által írt könyvből vettük [KW 94]. A feladat megadásában az első feltétel az elsőlépcsős feladat, míg a második két sor alkotja a második lépcsős pótlási feladatot.

$$
\begin{array}{rrrrrrrrl}
 & \min(2x_1 & + & 3x_2 & +E(7y_1 & +12y_2) & & & \\
\text{f.h.} & x_1 & + & x_2 & & & \leq & 100 & \\
 & (2+\eta_1)x_1 & + & 6x_2 & +y_1 & & \geq & 180+\zeta_1 & \\
 & 3x_1 & + & (3.4-\eta_2)x_2 & & +y_2 & \geq & 162+\zeta_2, &
\end{array}
$$

Itt a ζ_1, ζ_2 egy-egy csonkolt normális eloszlást követnek, η_1 egyenletes eloszlású, η_2 pedig egy csonkolt exponenciális eloszlású valószínűségi változó

$$
\begin{aligned}
\zeta_1 \in \mathrm{N}(0,12), &\quad \zeta_1 \in [-30.91, +30.91], \\
\zeta_2 \in \mathrm{N}(0,9), &\quad \zeta_1 \in [-23.18, +23.18], \\
\eta_1 \in \mathrm{U}(-0.8, +0.8), &\quad \eta_1 \in [-0.8, +0.8], \\
\eta_2 \in \mathrm{Exp}(\lambda = 2.5), &\quad \eta_2 \in (0.0, 1.84).
\end{aligned}
$$

A publikált legáltalánosabb változat az, amikor az összes valószínűségi változó szerepel, és $5 \times 9 \times 7 \times 11 = 3465$ diszkretizálást használtak a szerzők a $\eta_1, \eta_2, \zeta_1, \zeta_2$ valószínűségi változókra. Az általuk adott optimális megoldást $\mathbf{x}_D$-vel jelöljük, és a diszkretizált feladat első lépcsős költségét $f(\mathbf{x}_D)$ adja:

$$
\mathbf{x}_D = (37.754, 23.629), f(\mathbf{x}_D) = 150.446, f_{first}(\mathbf{x}_D) = 146.396.
$$

Az SRA algoritmus által elért eredményeinket a 6.6 Táblázatban adjuk meg.

	$\mathbf{x}$	$\mathbf{c}'\mathbf{x} + E(\mathbf{q}'\mathbf{y})$	"pontos"	$M =$	iter.szám
$discr_{3465}$	(37.75, 23.63)	150.446	151.06(± 0.12)		
SRA	(37.73, 27.10)	159.3(± 2.0)	157.6(±0.7)	100	100
SRA	(42.17, 26.10)	164.0(±0.4)	163.5(±0.1)	100	100
SRA	(37.68, 24.88)	150.6(±0.4)	152.5(±0.2)	400	100
SRA	(37.73, 27.10)	149.7(±1.6)	151.8(±0.2)	100	400

6.6. táblázat. Az SRA eredményei a Kall-Wallace példára, különböző mintaszámokra, kezdeti halmazokra és iteráció-számra.

A megadott $\mathbf{x}_D$ optimális megoldást ellenőriztük azzal, hogy kiszámítottuk ezen a helyen a költségfüggvényt, amelyben a következő numerikus értéket kaptuk:

$$\tilde{f}(\mathbf{x}_D) = 151.06(\pm 0.12) = 146.40 + 4.66(\pm 0.12).$$

A $|\tilde{f}(\mathbf{x}_D) - f(\mathbf{x}_D)| = 0.61$ különbség a diszkretizálás hibáját mutatja, hiszen ez szignifikánsan nagyobb, mint az eredményünk 0.12 nagyságú szórása (illetőleg ennek háromszorosa.

6.4. Vegyes feladat: kétlépcsős feladat valószínűségi korláttal

6.4.1. A vegyes feladat felépítése

A kétlépcsős feladat egyik hiányossága, hogy a második lépcsős lineáris programozási feladatnak nem minden $(\mathbf{x}, \boldsymbol{\xi})$ vektorpár esetén van megengedett megoldása (vagy véges optimuma). Ezt általában a teljes (vagy relative teljes) pótlás feltételezésével szokták kezelni, vagy pedig az úgynevezett indukált feltételek bevezetésével.

Prékopa javasolt egy modellt ([Pr 95], p. 417-418.) ennek a hiányosságnak egy másmilyen kezelésére: ez a kétlépcsős modell és a valószínűségi korlátos modell jellemzőinek együttes felhasználását igényli. Ezen feladatnak az a lényege, hogy a második lépcsős feladat megoldhatóságát csak egy $p > 0$ valószínűséggel követeli meg.

Tekintsük az eredeti második lépcsős feladatot, amelynek a formája

$$\begin{aligned} \mathcal{Q}(\mathbf{x}, \boldsymbol{\xi}) &= \min_{\mathbf{y}} \mathbf{q}'\mathbf{y} \\ \text{f.h. } T\mathbf{x} + W\mathbf{y} &= \boldsymbol{\xi}, \\ \mathbf{y} &\geq 0. \end{aligned} \tag{6.20}$$

Ez a feladat akkor és csak akkor oldható meg egy bizonyos $(\mathbf{x}, \boldsymbol{\xi})$ vektorpárra, ha a

$$\begin{aligned} \min_{\mathbf{y}} \quad & \mathbf{0}'\mathbf{y} \\ \text{f.h. } W\mathbf{y} &= \boldsymbol{\xi} - T\mathbf{x}, \\ \mathbf{y} &\geq 0 \end{aligned}$$

feladatnak van megengedett megoldása. A dualitás tétele miatt ez ekvivalens az előbbi feladat duáljának megoldhatóságával (és véges optimumának létezésével), vagyis a

$$\begin{aligned} &\max_{\mathbf{u}} (\boldsymbol{\xi} - T\mathbf{x})'\mathbf{u}, \\ &\text{f.h. } W'\mathbf{u} \leq 0. \end{aligned}$$

feladat megoldhatóságával. Jelölje az $\{\mathbf{u} \mid W'\mathbf{u} \leq 0\}$ kúp extremális irányainak halmazát $U = \{\mathbf{u}^{(1)}, \mathbf{u}^{(2)}, \ldots, \mathbf{u}^{(N)}\}$. A $(\boldsymbol{\xi} - T\mathbf{x})'\mathbf{u}^{(i)}$ függvény értéke nem lehet pozitív egyetlen $i = 1, \ldots, N$ index esetén sem, ugyanis ekkor a duál célfüggvény értéke végtelen lenne. Tehát ebből következően a $(\boldsymbol{\xi} - T\mathbf{x})'U \leq \mathbf{0}$ egyenlőtlenségnek (minden komponensben) teljesülnie kell, és ez az egyenlőtlenség ekvivalens a primál feladat (az eredeti második lépcsős feladat) megoldhatóságával. Összegezve mondhatjuk, hogy a második lépcsős feladatnak akkor és csak akkor van megengedett megoldása és véges optimuma, ha az

$$U'\boldsymbol{\xi} \leq U'T\mathbf{x} \tag{6.21}$$

egyenlőtlenségrendszer fennáll. A vegyes modellben ennek a feltételnek a teljesülését kívánjuk meg egy előírt $p > 0$ valószínűséggel. Ha a $\boldsymbol{\xi}$ valószínűségi vektorváltozó eloszlásfüggvénye logkonkáv, akkor a $Pr\{U'\boldsymbol{\xi} \leq U'T\mathbf{x}\}$ valószínűségi korlát is logkonkáv, tehát a megengedett megoldások halmaza konvex.

Ezek alapján a vegyes feladatnak az első lépcsője a következő formába irható:

$$\begin{aligned} \min \mathbf{c}'\mathbf{x} \quad &+ \quad E(\mathcal{Q}(\mathbf{x}, \boldsymbol{\xi})), \\ \text{f.h. } A\mathbf{x} \quad &\leq \quad \mathbf{b}, \\ Pr\{U'\boldsymbol{\xi} \geq U'T\mathbf{x}\} \quad &\geq \quad p, \\ \mathbf{x} \quad &\geq \quad 0, \end{aligned} \tag{6.22}$$

ahol a $\mathcal{Q}(\mathbf{x}, \boldsymbol{\xi})$ függvényt úgy definiáljuk mint a

$$\begin{aligned} \mathcal{Q}(\mathbf{x}, \boldsymbol{\xi}) &= \min_{\mathbf{y}, \mathbf{z}^+, \mathbf{z}^-} \mathbf{q}'\mathbf{y} + \mathbf{d}^{+'}\mathbf{z}^+ + \mathbf{d}^{-'}\mathbf{z}^- \\ \text{f.h. } W\mathbf{y} + \mathbf{z}^+ - \mathbf{z}^- &= \boldsymbol{\xi} - T\mathbf{x}, \\ \mathbf{y}, \mathbf{z}^+, \mathbf{z}^- &\geq 0, \end{aligned} \tag{6.23}$$

második lépcsős feladat optimális célfüggvény értékét. Itt a $\mathbf{z}$ változóra a szokásos módon bevezettük a $\mathbf{z} = \mathbf{z}^+ - \mathbf{z}^-$ nemnegatív tagú összegre való felbontását, miáltal lehetővé

válik a megengedettségtől való pozitív, illetőleg negatív irányú eltérést különböző $\mathbf{d}^{+'}$ illetőleg $\mathbf{d}^{-'}$ költségekkel való büntetése. Ebben a feladattípusban a $\mathbf{d}^+, \mathbf{d}^- \geq 0$ egyenlőtlenségeknek mindig fenn kell állnia. Természetesen, ha egy adott $(\mathbf{x}, \boldsymbol{\xi})$ vektorpárra az eredeti (6.20) második lépcsős feladatnak van megengedett megoldása, akkor azt kell megoldani és $\mathbf{z} = \mathbf{0}$ értéket kell helyettesíteni.

6.4.2. *SRA* a vegyes feladatra

A (6.22) és a (6.23) alatt megadott vegyes feladatra a már szokásos módon alkalmazhatjuk a szukcesszív regressziós approximációk eljárását. Mindkét nehezen kiszámítható függvényt, a $E\mathcal{Q}(\mathbf{x}, \boldsymbol{\xi})$ várható pótlást és a $Pr\{U'\boldsymbol{\xi} \leq U'T\mathbf{x}\}$ valószínűségi korlátot helyettesítjük a $q_k(\mathbf{x})$ illetőleg az $f_k(\mathbf{x})$ regressziós függvénnyel. A várható pótlás értékeinek kiszámítása az előző 6.2 szakaszban leírtak alapján végezhető el. Adott $\mathbf{x}_k$ és egy $\boldsymbol{\xi}_{kj}$ minta esetén q_{kj} függvényérték a

$$\begin{aligned} q_{kj}(\mathbf{x}_k) &= \min \mathbf{q}'\mathbf{y} + \mathbf{d}^{+'}\mathbf{z}^+ + \mathbf{d}^{-'}\mathbf{z}^- \\ W\mathbf{y} + \mathbf{z}^+ - \mathbf{z}^- &= \boldsymbol{\xi}_{kj} - T\mathbf{x}_k, \\ \mathbf{y}, \mathbf{z}^+, \mathbf{z}^- &\geq 0 \end{aligned} \tag{6.24}$$

lineáris programozási feladat optimális célfüggvényértéke, amelyet $j = 1, \ldots, s_q$ számú mintára határozunk meg. A $Q(\mathbf{x}_k)$ függvényérték egy torzítatlan becslése az

$$\frac{1}{s_q} \sum_{j=1}^{s_q} q_{kj}$$

átlag. A valószínűségi korlát függvényértékeinek kiszámítását például a durva Monte Carlo módszer segítségével végezhetjük el. Tekintsük a $\boldsymbol{\xi}$ valószínűségi változó $\boldsymbol{\xi}_{ki}, i = 1, 2, \ldots, s_p$ független mintáit, és definiáljuk a χ_{ki} valószínűségi változó lehetséges értékeit az alábbi módon:

$$\chi_{ki} = \begin{cases} 1, & \text{ha } \{U'\boldsymbol{\xi}_{ki} \leq U'T\mathbf{x}_k\} \text{ teljesül,} \\ 0, & \text{egyébként.} \end{cases} \tag{6.25}$$

Ezek után az

$$f_k = \frac{1}{s_p} \sum_{i=1}^{s_p} \chi_{ki}$$

átlag a torzítatlan becslése az $f(\mathbf{x}_k) = Pr\{U'\boldsymbol{\xi} \leq U'T\mathbf{x}_k\}$ függvényértéknek.

Mindkét zajos függvényérték szórása függ attól a ponttól, amelyben a függvényértéket becsültük, de az előzőkhöz hasonlóan a szórásokat állandóknak (de egymástól különbözőknek) tesszük fel. Az *SRA* algoritmus elindítása előtt szükségünk van egy $\mathbf{x}_0, \mathbf{x}_1, \ldots, \mathbf{x}_{k-1}$ ponthalmazra, valamint ezekben a két függvény zajos értékeire, vagyis feltesszük, hogy a

rendelkezésünkre áll az $S_k = \{\mathbf{x}_i, q_i\}_{i=0}^{k-1}$, és a $P_k = \{\mathbf{x}_i, f_i\}_{i=0}^{k-1}$ pont-függvényérték halmaz.

Az SRA algoritmus - a vegyes feladatra

0. Legyen a k iterációs számláló egyenlő az S_k, P_k halmazokban lévő pontok száma.
1. Határozzuk meg a $q_k(x) = \mathbf{x}'D_k\mathbf{x} + \mathbf{b}_k'\mathbf{x} + c_k$ regressziós függvény $D_k, \mathbf{b}_k, c_k$ paramétereit az S_k halmazból és az $f_k(x) = -\mathbf{x}'F_k\mathbf{x} + \mathbf{e}_k'\mathbf{x} + h_k$ regressziós függvény $F_k, \mathbf{e}_k, h_k$ együtthatóit a P_k halmazból.
2. Helyettesítsük az eredeti első lépcsős feladatot a

$$\begin{aligned} \min_{\mathbf{x}} \mathbf{c}'\mathbf{x} + q_k(\mathbf{x}) \\ A\mathbf{x} = \mathbf{b}, \\ f_k(\mathbf{x}) \geq p, \\ \mathbf{x} \geq 0 \end{aligned} \tag{6.26}$$

közelítő feladattal és jelöljük ennek egy optimális megoldását $\mathbf{x}_k$-val.
3. Ha $\mathbf{x}_k$ is ,,elég jó'', akkor STOP, egyébként generáljuk az $\boldsymbol{\xi}_{ki}$ mintákat és számítsuk ki a $q_k \sim \mathcal{Q}(\mathbf{x}_k, \boldsymbol{\xi})$ és az $f_k \sim Pr\{U'\boldsymbol{\xi} \geq U'T\mathbf{x}_k\}$ függvényértékeket a fentebb leírt módon. Adjuk hozzá az új pontot és a q_k, f_k függvényértékeket az S_k és a P_k halmazokhoz: legyen $S_{k+1} = S_k \cup \{\mathbf{x}_k, q_k\}$, $P_{k+1} = P_k \cup \{\mathbf{x}_k, f_k\}$. Növeljük meg az iterációs számlálót $k = k + 1$ és menjünk vissza az 1.lépésre.

Természetesen mindkét zajos függvény értékeinek kiszámításánál használhatunk másmilyen módszert is. Valószínűségek kiszámításának esetén ilyen eljárásokat írtunk le a 3. fejezetben, a várható pótlás értékének meghatározására pedig egy hatékony becsléscsaládot 6.5 szakaszban közlünk.

6.4.3. A vegyes feladat egy numerikus példája

Ezt a numerikus feladatot az előző részben megadott kétlépcsős feladatból származtattuk: eltöröltük a W mátrix első oszlopát. Így legyen most

$$W = \begin{pmatrix} -0.7 & 1.7 \\ 9.0 & -13.0 \end{pmatrix},$$

ezzel a W mátrixszal a (6.20) második lépcsős feladatnak nem minden esetben lett megengedett megoldása. A $W'\mathbf{u} \leq 0$ kúp extremális irányai az $\mathbf{u}^{(1)'} = (-9.0, -0.7), \mathbf{u}^{(2)'} = (-13.0, -1.7)$. Az eredeti második lépcsős és első lépcsős feladatok többi paraméterét nem változtattuk meg. Az új, bevezetésre kerülő valószínűségi korlátban szereplő mátrixok a következők:

$p=$	$\mathbf{x}$	$\mathbf{c}'\mathbf{x}+E(\mathbf{q}'\mathbf{y})$	$Pr\{\cdot\}$
0.70	(0.8366, 0.0238)	30.91($\pm$1.70)	0.7200($\pm$0.030)
		31.75($\pm$0.20)	0.7002($\pm$0.005)
0.80	(0.8040, 0.0491)	37.56($\pm$2.00)	0.8300($\pm$0.040)
		38.09($\pm$0.22)	0.8046($\pm$0.004)
0.90	(0.7716 0.0830)	47.53($\pm$2.30)	0.9300($\pm$0.030)
		47.82($\pm$0.20)	0.9019($\pm$0.003)
0.95	(0.7432, 0.1127)	57.77($\pm$2.40)	0.9800($\pm$0.010)
		57.69($\pm$0.30)	0.9539($\pm$0.002)

6.7. táblázat. Vegyes feladat: SRA eredmények különböző p megbízhatóságokra, $M = 100$, $k = 1000$, a költségvektor $\mathbf{d}^1$.

$$U' = \begin{pmatrix} -9.0 & -0.7 \\ -13.0 & -1.7 \end{pmatrix}, \quad U'T = \begin{pmatrix} -49.53 & 83.64 \\ -67.29 & 121.64 \end{pmatrix}, \tag{6.27}$$

A megengedett megoldástól való $\mathbf{z} = \mathbf{z}^+ - \mathbf{z}^-, \mathbf{z}^+ \geq 0, \mathbf{z}^- \geq 0$ eltérést a következő költségvektorokkal büntettük: $\mathbf{d}^1 = (9.0, 10.0)$, $\mathbf{d}^2 = (2.0, 10.0)$, $\mathbf{d}^3 = (10.0, 2.0)$, $\mathbf{d}^4 = (30.0, 20.0)$. Minden feladatunkban azonos költséggel büntettük a megengedettségtől való pozitív és negatív irányú eltérést, vagyis $\mathbf{d}^i = \mathbf{d}^{i,+} = \mathbf{d}^{i,-}$ volt.

Az erre a vegyes feladatra vonatkozó számítógépes eredményeket a 6.7, 6.8 és 6.9 táblázatokban közöljük. A 6.7 és a 6.8 táblázatokban a második sor minden példavariánsra a „pontos" eredményeket tartalmazza. A 6.8 táblázatban különböző kiindulásul vett ponthalmazok és költségvektorok hatását mutatjuk be.

A 6.9 táblázatban pedig egy számítógépes futásnak a k-adik lépésében elért eredményeit mutatjuk be, kiemelve azt, hogy hogyan változik a közelítő optimális megoldás. A $\mathbf{c}'\mathbf{x} + E(\mathbf{q}'\mathbf{y})$ oszlopban az adott $\mathbf{x}_k$ esetén meghatározott zajos függvényértékeket adtuk meg, míg a mellette álló $q_k(\mathbf{x})$ fejlécű oszlopban az aktuális regressziós közelítésnek az értékét adjuk meg. Hasonlóképpen a $Pr\{\cdot\}$ az adott pontban becsült függvényérték szerepel, míg az utolsó P_{app} oszlopban a regressziós közelítésen felvett értéket mutatjuk be. Látszik, hogy mind a két közelítő függvény lassabb, de határozott konvergenciát mutat (az optimális célfüggvényértékhez, illetőleg az előírt valószínűséghez). Végül a táblázat alatti szövegben szereplő $D^2(f(\cdot))$ érték az egy célfüggvényérték kiszámításában fellépő szórás, illetőleg a $D^2(P\{\cdot\})$ a valószínűségi korlát kiszámításának szórása.

	$\mathbf{x}$	$\mathbf{c}'\mathbf{x} + E(\mathbf{q}'\mathbf{y})$	$Pr\{\cdot\}$	$M =$	$k =$
$\mathbf{d}^1$	(0.768, 0.087)	50.96($\pm$2.3)	0.920($\pm$0.030)	100	100
		49.44($\pm$0.23)	0.908($\pm$0.003)		
$\mathbf{d}^2$	(0.768, 0.087)	50.75($\pm$2.3)	0.921($\pm$0.030)	100	100
		49.18($\pm$0.26)	0.909($\pm$0.003)		
$\mathbf{d}^3$	(0.768, 0.086)	50.76($\pm$2.3)	0.918($\pm$0.030)	100	100
		49.14($\pm$0.25)	0.906($\pm$0.003)		
$\mathbf{d}^4$	(0.770, 0.084)	51.03($\pm$2.3)	0.930($\pm$0.030)	100	500
		49.10($\pm$0.24)	0.898($\pm$0.003)		
$\mathbf{d}^1$	(0.767, 0.088)	51.31($\pm$2.3)	0.922($\pm$0.030)	100	100
		49.78($\pm$0.23)	0.911($\pm$0.003)		
$\mathbf{d}^1$	(0.769, 0.085)	47.33($\pm$1.2)	0.933($\pm$0.019)	400	400
		49.07($\pm$0.25)	0.907($\pm$0.003)		
$\mathbf{d}^1$	(0.773, 0.081)	51.87($\pm$2.6)	0.920($\pm$0.030)	100	5000
		47.72($\pm$0.06)	0.898($\pm$0.001)		

6.8. táblázat. Vegyes feladat: az *SRA* eredményei különböző költségvektorokra, kezdeti halmazokra, mintaszámokra és iterációszámra, a megengedettség $p = 0.9$ volt.

$k =$	$\mathbf{x}$	$\mathbf{c}'\mathbf{x} + E(\mathbf{q}'\mathbf{y})$	$q_k(\mathbf{x})$	$Pr\{\cdot\}$	P_{app}
1	(0.7741, 0.0804)	47.01	45.44	0.940	0.89
10	(0.7677, 0.0871)	51.07	50.13	0.930	0.90
100	(0.7677, 0.0871)	45.94	49.54	0.860	0.90
1000	(0.7708, 0.0838)	46.87	48.55	0.860	0.90
4000	(0.7722, 0.0823)	45.88	48.29	0.910	0.90
9999	(0.7718, 0.0828)	51.54	48.03	0.910	0.90

6.9. táblázat. Vegyes feladat: az *SRA* előrehaladása, költségvektor $\mathbf{d}^1$, mintaszám $M = 100$, $D^2(f(\cdot)) = 2.3^2$, $D^2(P\{\cdot\}) = 0.03^2$. „Pontos” értékek az utolsó pontban $f(\cdot) = 48.18(\pm 0.08)$, $P\{\cdot\} = 0.9005(\pm\ 0.0009)$.

6.5. A várható pótlás függvényének kiszámítása Monte Carlo integrálással

Ebben a részben Monte Carlo integrálási eljárásokat közlünk, amelyekkel számítástechnikailag hatékony módon meghatározhatók a várható pótlás függvényének értékei, ha a második lépcsős feladatban szereplő $\boldsymbol{\xi}$ valószínűségi vektorváltozó többdimenziós normális eloszlású.

Az eljárások arra az esetre is működnek, amikor a valószínűségi változó komponensei korreláltak, sőt, a numerikus hatékonyság lényegében azonos független és korrelált komponensek esetére. Az általunk követett eljárások bizonyos vonásait más eloszlásokra is használni lehet. Bár az ilyen normális eloszlású, korrelált komponensű $\boldsymbol{\xi}$ valószínűségi vektorváltozót tartalmazó második lépcsős feladatot rögzített T mátrix esetén egy lineáris transzformációval át lehet vinni egy ekvivalens feladatba, amelyben a jobboldali véletlen vektor független komponensű, de ez csak rögzített T mátrix (fixed recourse) esetén valósítható meg bonyodalmak nélkül. Ha a T mátrix véletlen komponenseket is tartalmaz (amelyek eloszlása a normálistól különböző is lehet), akkor a lineáris transzformáció után más, véletlen változók jelennek meg (az eredeti változók összegei), amelyek kezelése kérdéses.

Hasonló munkát végeztek Shapiro és de Mello [SH 98], de az általuk követett eljárásban a második lépcső duáljának célfüggvényértékét határozták meg, továbbá ellenőrző (control) változókat használtak – amivel csak az egyes pontokban felvett függvényértékek összehasonlítása lett könnyebb, de a függvényérték nem lett pontosabb. Nem foglalkoztak a korrelált esettel és nem lehet tudni, hogy milyen gyors az eljárásuk, mert számítógépes futási időket nem közöltek cikkükben.

Megismételjük a feladat leírását. Legyen a várható pótlás függvénye:

$$Q(\mathbf{x}) = E(\mathcal{Q}(\mathbf{x}, \boldsymbol{\xi})) = E(\min_{\mathbf{y}} \mathbf{q}'\mathbf{y} | W\mathbf{y} = \boldsymbol{\xi} - T\mathbf{x}, \mathbf{y} \geq 0). \tag{6.28}$$

Ebben a szakaszban torzítatlan Monte Carlo becsléseket adunk a $Q(\mathbf{x})$ függvényértékek kiszámítására, ha $\boldsymbol{\xi}$ többdimenziós normális eloszlású. Becsléseink megalkotásában részben az eloszlásfüggvény kiszámítására alkalmazott dekompozíciót és ortogonalizált becslések ötletét, részben pedig a Monte Carlo módszerekben szokásos rétegezett mintavételt és az ellentétes (antithetic) változók módszerét [De 90a], [HH 64] használtuk. A numerikus hatékonyságot ezen gondolatok megvalósításának keverékeként sikerült elérni.

Ebben a szakaszban a normális eloszlású vektor dimenzióját n jelöli, eloszlásfüggvényét Φ, $\boldsymbol{\xi}$ komponenseinek várható értéke 0, szórása 1, a korrelációs mátrixa pedig C. Mint ismert, egy R alsó-háromszög mátrix segítségével felírható a $C = RR'$ egyenlőség.

Megjegyezzük, hogy a $Q(\mathbf{x})$ várható pótlás függvénye konvex, hasonlóan a $\mathcal{Q}(\mathbf{x}, \boldsymbol{\xi})$ függvény is konvex mindkét változójában (lásd például [Pr 95], [RSch 87]). $Q(\mathbf{x})$ értelmezési tartománya konvex poliéderek egyesítése, és egy-egy ilyen poliéder felett a pótló függvény egy lineáris függvény (az ilyen függvényeket röviden szakaszonként lineárisnak mondjuk). Továbbá ha $\mathbf{x}$ elég „messze" van az optimális megoldástól, $\mathcal{Q}(\mathbf{x}, \boldsymbol{\xi})$ függvény a $\boldsymbol{\xi}$ változóban lineáris lesz (nagy valószínűséggel).

Ezt a következőképpen lehet belátni. Egyszerűség kedvéért tegyük fel, hogy $E\boldsymbol{\xi} = 0$ és $Q(\mathbf{x})$ az egész téren értelmezve van. A $\mathcal{Q}$ pótló függvény értéke a $W\mathbf{y} = \boldsymbol{\xi} - T\mathbf{x}, \mathbf{y} \geq 0$ feltételek melletti feladat $\min \mathbf{q}'\mathbf{y}$ optimális célfüggvényértéke. Ha $\mathbf{x}$ és ezzel $-T\mathbf{x}$ is elég nagy, akkor ugyanazok a $B_i, i = 1, \ldots, s$ bázisok lesznek megengedettek a $-T\mathbf{x}$ és a $\boldsymbol{\xi} - T\mathbf{x}$ jobboldalak esetén is (nagy valószínűséggel). Ezeken a bázisokon a célfüggvényértékek az elsőnek adott jobboldali vektor esetén nyilván $\mathbf{q}'\mathbf{y}_i = \mathbf{q}'B_i^{-1}(-T\mathbf{x})$ (illetőleg $\min \mathbf{q}'\mathbf{y}_i = \mathbf{q}'B_i^{-1}(\boldsymbol{\xi} - T\mathbf{x})$). Legyen az elsőnek adott jobboldal esetén az optimális bázis B_1, a célfüggvényérték f_1. Mivel csak véges sok megengedett bázis van, így elég nagy $\mathbf{q}'B_i^{-1}(-T\mathbf{x})$ érték esetén az f_1 célfüggvényérték sem változhat meg annyira a második megadott jobboldal esetén, hogy az eddig optimális B_1 bázis ne maradjon optimális – egy elég nagy $-T\mathbf{x}$ jobboldali vektor $\boldsymbol{\xi}$ vektorral való perturbálása nem változtatja meg a B_1 bázis optimális bázis tulajdonságát (nagy valószínűséggel). Ha pedig az optimális bázis ugyanaz a $\boldsymbol{\xi}$ különböző értékeire, akkor a $\mathcal{Q}(\mathbf{x}, \boldsymbol{\xi})$ függvény elég nagy $\mathbf{x}$ érték esetén a $\boldsymbol{\xi}$ változóban valóban lineáris lesz.

Tetszőleges $\mathbf{x}$ pont és a $\boldsymbol{\xi}$ valószínűségi változó $\boldsymbol{\xi}_i$ realizációja esetén megoldható a

$$\begin{aligned} q_i = \mathcal{Q}(\mathbf{x}, \boldsymbol{\xi}_i) &= \min_{\mathbf{y}} \mathbf{q}'\mathbf{y} \\ \text{f.h. } T\mathbf{x} + W\mathbf{y} &= \boldsymbol{\xi}_i, \\ \mathbf{y} &\geq 0, \end{aligned} \tag{6.29}$$

feladat, melynek optimális q_i célfüggvényértékére $Eq_i = Q(\mathbf{x})$ igaz. Ezek szerint ha a $\boldsymbol{\xi}$ valószínűségi vektorváltozónak $\boldsymbol{\xi}_i, i = 1, 2, \ldots, N$ független realizációi, akkor a

$$\Theta_1 = \frac{1}{N} \sum_{i=1}^{N} \mathcal{Q}(\mathbf{x}, \boldsymbol{\xi}_i)$$

átlag a $Q(\mathbf{x})$ egy torzítatlan becslése, ezt a durva becslésnek nevezzük. Egy q_i függvényérték meghatározása egy lineáris programozási feladat megoldását jelenti, ami időigényes lehet, ezért a továbbiakban olyan becsléseket konstruálunk, amelyekben a becslés szórása csökken (bár a $\mathcal{Q}$ függvény kiszámítások száma nőhet).

6.5.1. Dekompozíció

Tekintsük a normális eloszlású $\boldsymbol{\xi}$ valószínűségi változó

$$\boldsymbol{\xi} = \chi_n R\boldsymbol{\eta}$$

alakú dekompozícióját, ahol χ_n egy n-szabadságfokú χ-eloszlású valószínűségi változó, $F_n(k)$ eloszlásfüggvénnyel, $\boldsymbol{\eta}$ pedig egy, az $S_n = \{\mathbf{x} | \sum_{i=1}^{n} x_i^2 = 1\}$ egységgömb felületén egyenletes eloszlású vektor, melynek eloszlásfüggvénye $V_n(\mathbf{y})$ (az egységgömb felületén megadott Lebesgue mérték egyre normálva). Megjegyezzük, hogy χ_n és $\boldsymbol{\eta}$ függetlenek. A $Q(\mathbf{x}) = E\mathcal{Q}(\mathbf{x}, \boldsymbol{\xi}) = E\left[E\mathcal{Q}(\mathbf{x}, \chi_n R\boldsymbol{\eta}) | \boldsymbol{\eta}\right]$ egyenlőség segítségével felírhatjuk, hogy

$$Q(\mathbf{x}) = \int_{R^n} \mathcal{Q}(\mathbf{x}, \boldsymbol{\xi}) d\Phi(\boldsymbol{\xi}) = \int_{S_n} \int_0^\infty \mathcal{Q}(\mathbf{x}, kR\mathbf{y}) dF_n(k) dV_n(\mathbf{y}). \tag{6.30}$$

Így integrálunkat egy egydimenziós és egy $(n-1)$-dimenziós integrálra bontottuk. Ha u a $[0,1)$ intervallumban egyenletes eloszlású valószínűségi változó, akkor az $F_n^{-1}(u)$ valószínűségi változó eloszlásfüggvénye $F_n(\cdot)$ a standard inverziós véletlenszámgenerálási módszer szerint. Ezt felhasználva

$$Q(\mathbf{x}) = \int_{S_n} \int_0^1 \mathcal{Q}(\mathbf{x}, F_n^{-1}(u) R\mathbf{y}) du dV_n(\mathbf{y}). \tag{6.31}$$

Ha most az $\boldsymbol{\eta}$ és az u valószínűségi változók független realizációit $\mathbf{y}_i, i = 1, 2, \ldots, N_1$ illetőleg $u_j, j = 1, 2, \ldots, N_2$ jelöli, akkor megkonstruálhatjuk a következő

$$\Theta_2 \quad = \quad \frac{1}{N_1 N_2} \sum_{i=1}^{N_1} \sum_{j=1}^{N_2} \mathcal{Q}(\mathbf{x}, F_n^{-1}(u_j) R\mathbf{y}_i)$$

becslést. A becslésnek az a lényege, hogy az S_n felületen egyenletes eloszlású irányokat generálunk és mindegyik irányon végrehajtunk egy (közelítő) integrálást. Először az utóbbival foglalkozunk, megmutatjuk hogyan becsülhető ez az egydimenziós integrál, és utána a külső, $(n-1)$-dimenziós integrál kiszámítására adunk meg becsléseket.

6.5.2. Vonal menti integrálás

Mindenekelőtt szimmetrizáljuk az adott irányon felhasznált pontokat. Legyenek a $\boldsymbol{\xi}$ független realizációi $\boldsymbol{\xi}_i$, akkor definiáljuk a következő becslést:

$$\Theta_3 = \frac{1}{N} \sum_{i=1}^{N} g_3(\mathbf{x}, \boldsymbol{\xi}_i), \quad g_3(\mathbf{x}, \boldsymbol{\xi}) = \frac{1}{2}[\mathcal{Q}(\mathbf{x}, \boldsymbol{\xi}) + \mathcal{Q}(\mathbf{x}, -\boldsymbol{\xi})].$$

A szimmetrizálás zéró (vagy nagyon kis) szórású becsléseket ad, ha az $\mathbf{x}$ pont „messze" van az optimális megoldások halmazától. Az ilyen és hasonló megjegyzések a számítógépes futások eredményei alapján adhatók meg, illetőleg a 6.5 pont utáni, a $\mathcal{Q}(\mathbf{x}, \boldsymbol{\xi})$ függvény természetére vonatkozó megjegyzések alapján – az optimumtól elég „messze" lévő $\mathbf{x}$ esetén a $\mathcal{Q}$ függvény már valóban lineáris a $\boldsymbol{\xi}$ változóban (nagy valószínűséggel), egy lineáris függvényre pedig $D^2(\Theta_3) = 0$ valóban. A Θ_3 becslés általában a $\mathcal{Q}(\cdot)$ függvény monotonitása miatt csökkenti a szórást, hiszen Θ_3 monoton függvényre az antitetikus változók módszerét alkalmazza (bár a konvexitás miatt csak az biztosítható, hogy egy monoton növő és egy monoton csökkenő részből áll a függvény).

A $\mathcal{Q}(\mathbf{x}, kR\mathbf{y})$ függvény adott $\mathbf{x}, \mathbf{y}$ esetén konvex a k változóban. Mivel $F_n^{-1}(\cdot)$ monoton növekvő, így $\mathcal{Q}(\mathbf{x}, F_n^{-1}(u)R\mathbf{y})$ mint az u függvénye egy monoton növekvő (vagy csökkenő) vagy pedig egy monoton növekvő és egy monoton csökkenő darabból áll (de ez a k-t

az $F_n^{-1}(u)$-val helyettesítő transzformáció a függvény konvexségét nem őrzi meg). Ezen monotonság miatt van értelme a következő becslésnek, amely egy antitetikus becslés a monoton függvények esetén [HH 64]:

$$\begin{aligned}\Theta_4 &= \frac{1}{N}\sum_{j=1}^{N} g_4(\mathbf{x}, u_j, \mathbf{y}),\\ g_4(\mathbf{x}, u, \mathbf{y}) &= 0.5\left[g_3(\mathbf{x}, F_n^{-1}(0.5u)R\mathbf{y}) + g_3(\mathbf{x}, F_n^{-1}(1-0.5u)R\mathbf{y})\right],\end{aligned}$$

ahol $\mathbf{y} \in S_n$ rögzített irány. Ha $\mathbf{x}$ közel van az optimumhoz, akkor sok esetben a $\mathcal{Q}$ függvény (majdnem) szimmetrikussá válik nemcsak az $\mathbf{x}$, hanem a $\boldsymbol{\xi}$ változóban is – tehát ilyen esetekben a szimmetrizálás csődöt mond: több számítási munkát kell végezni és semmilyen szóráscsökkenést nem érünk el. Az integrálási intervallumnak a $[0, \alpha]$ és az $[\alpha, 1)$ két részre való felbontása alapján egy rétegezett mintavétel válik lehetővé:

$$\begin{aligned}\Theta_5 &= \frac{1}{N}\sum_{j=1}^{N} g_5(\mathbf{x}, u_j, \mathbf{y}),\\ g_5(\mathbf{x}, u, \mathbf{y}) &= \alpha g_3\left(\mathbf{x}, F_n^{-1}(\alpha u R\mathbf{y})\right) + (1-\alpha) g_3\left(\mathbf{x}, F_n^{-1}\left(1-(1-\alpha)u\right)R\mathbf{y}\right).\end{aligned}$$

Az előző Θ_4 becslés az itt adott Θ_5 becslésnek az a speciális esete, amikor az $\alpha = 0.5$ felezőponti rétegezést használunk. Elvileg lehetőség van arra, hogy adott $\mathbf{x}, \mathbf{y}$ esetén meghatározzuk az α legjobb (legnagyobb szóráscsökkenést adó) értékét, de erről lemondunk – ugyanis adott $\mathbf{x}, \mathbf{y}$ esetén a $\mathcal{Q}$ függvénynek a hozzávetőleges alakját csak sok munkával lehetne meghatározni.

6.5.3. Ortogonalizálás

Most a (6.31)-ban szereplő integrálok közül a külső integrál meghatározásának feladatával foglalkozunk. Itt az integrálást az n-dimenziós térben elhelyezkedő S_n $(n-1)$-dimenziós felületen kell végrehajtani. Mivel a $\mathcal{Q}(\mathbf{x}, kR\boldsymbol{\eta})$ szórása adott k és változó $\boldsymbol{\eta}$ esetén nagy lehet, ezért „egyenletesebb" pontok használatát vezetjük be: egy egyenletes eloszlású $\boldsymbol{\eta}$ vektor helyett egy véletlen, ortonormalizált vektorrendszert használunk. Legyenek az $\mathbf{u}_i$ vektorok ortonormáltak: $\mathbf{u}_i\mathbf{u}_j = \delta_{ij}$, ahol $\delta_{ii} = 1, \delta_{ij} = 0, i \neq j, i, j = 1, 2, \ldots, n$ és legyen $\mathbf{U} = \{\mathbf{u}_i\}_{i=1}^n$ az ezekből álló, az S_n felett egyenletes eloszlású vektorrendszer. Ekkor az ortonormált becslés

$$\Theta_6 = \frac{1}{N}\sum_{i=1}^{N} g_6(\mathbf{x}, k, \mathbf{U}^{(i)}),\ g_6(\mathbf{x}, k, \mathbf{U}) = \frac{1}{n}\sum_{i=1}^{n}[\mathcal{Q}(\mathbf{x}, kR\mathbf{u}_i)],$$

ahol az $\mathbf{U} = \{\mathbf{u}_i\}_{i=1}^n$ vektorrendszer független realizációit $\mathbf{U}^{(i)}$ jelöli. Lényegében véve az ortonormalizált pontok és a dekompozíció együttes használata az eredeti durva becslésben a korreláció miatt fellépő szórás nagy részét eltünteti.

Az S_n felületen másmilyen „egyenletes” pontokat állíthatunk elő a következő meggondolások alapján: használhatjuk az $\{\mathbf{u}_i\}_{i=1}^n$ vektorok közül tetszőleges kettőnek a normalizált összegét. Így az $\mathbf{u}_i, i = 1, \ldots, n$ ortonormalizált rendszerben lévő n darab vektor helyett az $(\mathbf{u}_i + \mathbf{u}_j)/\sqrt{2}, i = 1, \ldots, n-1, j = i+1, \ldots, n$ alakú, $n(n-1)/2$ darab vektort használhatjuk. Az így konstruált pontokat használó becslést O_2 becslésnek nevezzük. Az O_2 becslések esetén szükséges számítási munkát tovább csökkenthetjük, ha észrevesszük, hogy az $R(\mathbf{u}_i + \mathbf{u}_j)/\sqrt{2}$ vektorok előállításának során az R mátrixszal való szorzást csak n-szer kell végrehajtani (az $R\mathbf{u}_i$ vektorok meghatározásához), nem pedig $n(n-1)/2$ alkalommal (lásd a 3. Fejezetben az ortonormált becsléseknél leírtakat).

6.5.4. A javasolt becslés

Összefoglalva a fentebbieket, valamint a számítógépes tapasztalatokat, a végsőként javasolt (és használt) becslés szimmetrizálást, felező-pontos rétegezést és ortogonalizálást használ. Tehát minden ortonormalizált vektor esetén négy pontot használunk az egydimenziós integrálás végrehajtására – ez az n darab vektor által adott irányra összesen $4n$ darab $\mathcal{Q}(\mathbf{x}, \boldsymbol{\xi})$ függvényérték kiszámítását igényli. A javasolt becslés a következő:

$$\begin{aligned}
\Theta_7 &= \frac{1}{N}\sum_{j=1}^{N} g_7(\mathbf{x}, u_j, \mathbf{U}^{(j)}), \\
g_7(\mathbf{x}, u, \mathbf{U}) &= \frac{1}{n}\frac{1}{4}\sum_{i=1}^{n} \mathcal{Q}(\mathbf{x}, F_n^{-1}(0.5u)R\mathbf{u}_i) + \mathcal{Q}(\mathbf{x}, -F_n^{-1}(0.5u)R\mathbf{u}_i) + \\
&\quad \mathcal{Q}(\mathbf{x}, F_n^{-1}(1.0-0.5u)R\mathbf{u}_i) + \mathcal{Q}(\mathbf{x}, -F_n^{-1}(1.0-0.5u)R\mathbf{u}_i).
\end{aligned} \tag{6.32}$$

Egy további lehetőségként említjük a következő becslést. A belső integrálra használjuk a szimmetrizálást és a felezőpontos rétegezést, míg a külső integrálra használjuk az előbb említett O_2 eljárást. Ezt a becslést jelölje Θ_8, ennek egy adott vektorrendszer és véletlen pont esetére összesen $4n(n-1)/2$ függvénymeghatározásra van szükség. A becslések összehasonlítása szerint ez a Θ_8 becslés általában legalább olyan jó volt, mint a Θ_7 becslés, de céljaink szempontjából ez túl pontos eredményeket adott (a sok függvénymeghatározás miatt lassú volt), ezért általában a Θ_7 becslés használata ajánlható.

6.5.5. Számítógépes eredmények

Becslések hatékonyságát két tényező, a számítási munka és a becslés szórása együttes figyelembevételével kell meghatározni [De 90a]. Mivel a becslésekben a $\mathcal{Q}(\mathbf{x}, \boldsymbol{\xi})$ függvényérték meghatározása szerepel (ami egy lineáris programozási feladat megoldását jelenti), és az összes többi számítási munka elhanyagolható ezzel összehasonlítva, ezért a számítási munkát a $\mathcal{Q}$ függvényértékek meghatározásának számával mérjük.

A Θ_i i-edik becslés hatékonyságát a durva módszerre vonatkoztatva számoljuk ki a

szokásos aránnyal:

$$\mathrm{Eff}_i \;=\; \frac{D^2(\Theta_1)}{D^2(\Theta_i)}\;\;\frac{1}{N_i},$$

ahol N_i a $\mathcal{Q}$ függvény kiszámításainak számát mutatja az i-edik becslés esetén (a durva módszernél csak egy darab $\mathcal{Q}$ függvényértéket kell kiszámítani), $D^2(\Theta_i)$ pedig az i-edik becslés szórásnégyzete. Az Eff_i arány azt mutatja, hogy ha N_i függvénymeghatározás árán az i-edik becslés egy adott szórású eredményt ad, akkor hányszor kell végrehajtani a durva becslést ahhoz, hogy az eredmények átlaga elérje ugyanazt a szórást.

Minden becslés szórásnégyzete, és így a hatékonysága is függ az $\mathbf{x}$ argumentum értékétől. A számítógépes futások eredményei alapján ha $\mathbf{x}$ „messze" van a feladat optimális megoldásától, akkor a függvény lineáris, vagy majdnem lineáris, így a szimmetrizálást használó eljárások szórása nagyon kicsi, vagy zéró lesz, a hatékonyság nagyon nagy lesz. Mivel a $\mathcal{Q}$ függvény az optimumhoz közel majdnem szimmetrikus szokott lenni, itt általában minden, a szimmetrizálásnak köszönhető hatékonyság eltűnik. Másrészt mivel a függvény alakjára nézve csak néhány tapasztalatunk van, és nem biztosítja semmi a függvény ilyen általános viselkedését (hiszen ha $\mathcal{Q}(\mathbf{x}, \boldsymbol{\xi})$ szimmetrikus $\mathbf{x}$-ben, abból nem következik a $\boldsymbol{\xi}$-ben való szimmetrikusság), ezért megtartjuk a szimmetrizálást a javasolt becslésben. A Θ_7 becslés másik előnye az, hogy a dekompozíció és az ortogonalizálás miatt a korrelált komponensű normális eloszlású vektorokra körülbelül ugyanolyan hatékony, mint a független komponensekre.

A legtöbb, kétlépcsős feladatra közölt megoldó algoritmus a folytonos eloszlású $\boldsymbol{\xi}$ valószínűségi vektorváltozót egy (véges tartományra csonkolt) diszkrét eloszlással helyettesíti [BL 97], [May 98]. Két probléma szokott ezzel kapcsolatban felmerülni: a dimenziószám és a korreláltság. Ezt a diszkretizálást erősen korrelált komponensekre, vagy 10-nél magasabb dimenzióban szinte lehetetlen kezelni. Továbbá, a diszkretizált eloszlásban szereplő pontok száma exponenciálisan nő a diszkretizálandó vektor dimenziószámának növekedésével, ha egy adott hibaszintet akarunk elérni. Ha a második lépcsős feladatban csak a jobboldali vektor véletlen, akkor a korrelált komponensű $\boldsymbol{\xi}$ vektor független komponensűvé transzformálható – azon az áron, hogy a második lépcső minden egyenlőtlenségét is transzformálni kell. Ha a második lépcső feladatában szereplő T mátrix rögzített (fixed recourse), akkor ez nem okoz nehézséget, viszont néhány véletlen komponenst is tartalmazó T mátrix esetén ez a véletlen hatás a lineáris transzformáció után az összes első lépcsős változó komponensében megjelenik, ráadásul konvolúciók formájában.

Numerikus eredményeinket, a becslések hatékonyságát egy, Kall és Mayer által megadott, Shapiro és de Mello [SH 98] által közölt numerikus példán szemléltetjük. Ebben a példában csak a jobboldali vektor volt véletlen, eloszlása pedig tízdimenziós normális eloszlású volt (a példa további leírását és megoldását lásd a következő 6.6 szakaszban).

Az optimális megoldáshoz közel az $N = 100$ mintapontot használó durva módszer esetén a becslés szórásnégyzete $D^2(\Theta_1)|_{N=100} = (0.09)^2$ volt. Az ajánlott Θ_7 becslést használva $N = 3$ mintaszámmal a becslés szórásnégyzete $D^2(\Theta_7)|_{N=3} = (0.03)^2$ volt – ezt az eredményt 0.1 sec alatt lehetett kiszámítani (ezalatt a 3 ortonormált rendszerből

	függetl., $f_{min} = 15.17$						Korrel., $\varrho = 0.95$, $f_{min} = 14.96$			
$f(\mathbf{x}) =$	15.17	15.18	15.40	16.30	23.11	74.13	14.97	15.16	17.27	28.74
Eff_3	2	1	5	1	∞	∞	1	2	499	∞
Eff_7	6	9	24	3	∞	∞	6	18	43	∞
Eff_8	9	8	8	4	∞	∞	3	3	8	∞

6.10. táblázat. A $\Theta_3, \Theta_7, \Theta_8$ becslések hatékonyságai, különböző $\mathbf{x}$ pontokban, független és korrelált komponensek esetére, f_{min} optimális célfüggvényérték.

$\mathbf{x}_{k,1}$	$f(\mathbf{x}_k)$	Hiba(σ)	Eff_7	Eff_8
0.5039	15.1802	$\pm$0.0010	38	8
0.5053	15.1886	$\pm$0.0005	22	7
0.5167	15.1757	$\pm$0.0004	6	4
0.5168	15.1749	$\pm$0.0010	26	6
0.5274	15.1753	$\pm$0.0003	8	15
0.5323	15.1791	$\pm$0.0005	5	4
0.5357	15.1784	$\pm$0.0003	9	7

6.11. táblázat. Hatékonyságok és célfüggvényértékek néhány, az *SRA* által előállított, optimum közeli pontban, független komponensek esete – az f célfüggvény minimuma 15.1752 volt.

származó pontokat használva 120 lineáris programozási feladatot oldottunk meg).

Az $f(\mathbf{x}) = \mathbf{c}'\mathbf{x}+Q(\mathbf{x})$ első lépcsős célfüggvény értékére Mayer az SLP-IOR programcsomag segítségével a következő korlátokat tudta megadni: $14.9927 < f(\mathbf{x}_{\mathbf{MAYER}}) < 15.6822$. Az általunk megadott Θ_7 becslés használatával egy ennél jóval pontosabb függvényértéket tudtunk kiszámítani a Mayer által megadott optimális pontban: $f(\mathbf{x}_{\mathbf{MAYER}}) \sim 15.175542, \pm 0.000026$.

Shapiro és de Mello az általuk használt módszer segítségével egy másik optimális pontot határoztak meg, amelyre a Monte Carlo módszerük segítségével kiszámított függvényértéket $f(\mathbf{x}_{\mathbf{SHAP-MELLO}}) = 15.1866$-re tették. A Θ_7 becslés használatával az általuk optimálisnak megadott pont esetén a függvényérték $f(\mathbf{x}_{\mathbf{SHAP-MELLO}}) \sim 15.175263, \pm 0.000025$ [SH 98]. Tehát a Θ_7 becslés használata nagyságrendekkel pontosabb eredményt adott, mint az eddig használt módszerek.

Az $f(\mathbf{x})$ függvényértékek után megadott $\pm$ mennyiségek most is a függvényérték szórását adják meg, nagy valószínűséggel ennek háromszorosa tekinthető a tényleges hiba egy korlátjának. Az egyszerűség kedvéért mindenhol a szórást adtuk meg, mint a hiba értékét, természetesen ez függ az $\mathbf{x}$ argumentumtól.

A számítógépes futások alapján a hatékonyságokkal kapcsolatban azt mondhatjuk, hogy minél kisebb volt a függvényérték, annál kisebbek voltak a hatékonyságok. Így az

alábbiakban különböző pontokra adjuk meg a hatékonyságokat, de többségükben ezek a legrosszabb eseteket tartalmazzák, amikor az $\mathbf{x}$ pont közel van az optimális megoldáshoz – a függvényérték pedig közel van a minimális függvényértékhez.

A 6.10 táblázatban néhány tipikus adatot közlünk a hatékonyságra. Az első hat oszlopban a független komponensekkel rendelkező véletlen vektort tartalmazó pótló függvény eseteit mutatjuk, az utolsó négy oszlopban pedig a korrelált komponensű véletlen vektorra mutatunk példákat – itt egy olyan korrelációs mátrixot használtunk, amelyben a korreláció minden változópárra ugyanaz volt: $\varrho = 0.95, c_{ij} = \varrho$ ha $i \neq j$ és $c_{ii} = 1$. A ∞ szimbólum a táblázatban azt jelenti, hogy az adott esetben a hatékonyság 1000-nél nagyobb volt, jól mutatva a függvény lineáris, vagy majdnem lineáris viselkedését.

Néhány egyéb esetben tapasztalt hatékonyságot adunk meg a 6.11 táblázatban – ezek a független komponensű véletlen vektorra vonatkoznak. Olyan pontokban számítottuk ki a hatékonyságot, amelyeket az *SRA* algoritmus 5-10 percnyi munka után adott ki közelítő optimális megoldásokként – ezeket a pontokat csak az első komponensükkel, az $\mathbf{x}_{k,1}$ értékekkel jeleztük a táblázatban. A második oszlopban az itt kiszámított függvényértékeket és a sejtésünk alapján számított szórásukat adtuk meg. Az utolsó két oszlop tartalmazza a Θ_7 és a Θ_8 becsléseknek a durva módszerhez viszonyított hatékonyságát.

A 6.7 szakaszban megadott, futási eredményeket tartalmazó 6.20 táblázat egy 20 dimenziós normális eloszlású vektor használata esetén tapasztalt hatékonyságnövekedést mutatja – a Θ_3 becslésről a Θ_7 becslésre való áttérés lényegesen gyorsítja az algoritmus futását.

6.6. A kétlépcsős feladat hatékony numerikus megoldása

Az eddigiek alapján a kétlépcsős sztochasztikus programozási feladat egy megoldását meg lehet adni az *SRA* algoritmus felhasználásával. Egy közepes méretű feladat elemzése kapcsán azonosítjuk a számítási eljárás numerikusan nehéz részleteit és megoldási eljárásokat adunk a felmerülő numerikus nehézségekre.

Az *SRA* alkalmazásával kapcsolatos numerikus megfontolásokat részben már érintettük a STABIL modell megtárgyalásánál; foglalkoztunk a kvadratikus regresszió meghatározásával, a kezdeti pontok számával és a pont-függvényhalmaz megadásával, a megállási kritériumok alkalmazásával. Ezeket értelemszerűen alkalmazni kell a kétlépcsős feladat megoldásánál is. Ez fordítva is igaz, az ebben a szakaszban leírtakat használni kell a nagyobb méretű valószínűséggel korlátozott feladatok, a 6.1. szakaszban megadott sztochasztikus kvadratikus optimalizálási alapfeladat numerikus megoldásánál, vagy a 6.7. szakasz nagyméretű feladatai esetén.

Az algoritmus gyorsítását lehet elérni azzal, hogyha a pótló függvény kiszámítására az előző szakaszban leírt Monte Carlo integrálási eljárásokat használjuk. A megfelelő numerikus stabilitás és hatékonyság eléréséhez még további módosításokat is alkalmazni kell. Ezeket a módosításokat írjuk le ebben a szakaszban. Ezek használata opcionális, kivéve

az alábbiakban ismertetendő degeneráció elleni védekezésre szolgáló eljárást, amelynek alkalmazására még kisméretű feladatok esetében is szükség van.

6.6.1. Numerikus stabilitás

A számítások numerikus stabilitásának elérése lényegében két, egymásnak ellentmondó cél megvalósítását jelenti. Az egyik cél az, hogy az S_k halmaz „elég széles" legyen azért, hogy az előállított $\mathbf{x}_i, i = 0, 1, \ldots, k$ pontok (illetőleg az általuk felvett függvényértékek) segítségével meghatározandó függvény egy pozitív definit mátrix által megadott kvadratikus függvény alakját vegye fel (kifeszítsen egy konvex függvényt). A másik célunk az, hogy a pontok nagy része lehetőleg az optimális megoldás (megoldások) egy szűk környezetében legyenek, hiszen ezáltal lesz a közelítésünk pontos az optimum környékén. A két cél együttes megvalósításának kulcsa abban rejlik, hogy néhány „messze lévő" pont elég a konvex függvény kifeszítéséhez, míg az összes többi pont lehet az optimum környékén.

A kezdeti ponthalmaz jó megadásának, a pontrendszer adaptív kiszélesítésének és a degeneráció elleni védekezésnek a célja az, hogy a ponthalmaz elég széles legyen (a közelítés stabil legyen), míg a legkisebb négyzetek módszerében alkalmazott súlyok használata azt eredményezi, hogy elég szűk lesz a ponthalmazban szereplő pontok nagy része által alkotott halmaz. Ebben a részben az $\mathbf{x}$ első lépcsős döntési változó dimenzióját egyszerűen n-nel jelöljük.

Kezdeti halmaz

Az algoritmus elindításához egy kezdeti ponthalmazra van szükségünk, ezt a halmazt például a következőképpen lehet előállítani. Tekintjük a kétlépcsős feladat definiálásánál használt determinisztikus alapfeladatot, ahol a $\boldsymbol{\xi}$ valószínűségi változót helyettesítettük a várható értékével. Tehát megoldjuk a

$$\begin{array}{rrcl} \min & \mathbf{c}'\mathbf{x} \quad +\mathbf{q}\mathbf{y} & & \\ \text{f.h.} & A\mathbf{x} & = & \mathbf{b}, \\ & T\mathbf{x} \quad +W\mathbf{y} & = & E\boldsymbol{\xi}, \\ & \mathbf{x}, \quad \mathbf{y} & \geq & 0, \end{array}$$

feladatot. Jelölje ennek az optimális megoldását $\mathbf{x}_0$, ezt nevezzük gyökérpontnak. A többi pontot a gyökérpont körül generáljuk véletlenszerűen: $\mathbf{x}_i = \mathbf{x}_0 + \delta\mathbf{d}_i, i = 1, \ldots, k_{in} - 1$, ahol δ tetszőleges (de a választás után rögzített), $0.1 - 0.5$ értékű volt, a $\mathbf{d}_i$ pontok pedig véletlenszerűen $+1$, vagy -1 értéket tartalmaztak minden komponensükben. Ezt az eljárást addig folytattuk, amíg a megfelelő k_{in} számú pontot nem állítottuk elő: az eljárás az esetek többségében megfelelő kezdeti halmazt állított elő. Nagyméretű feladataink esetén (6.7 szakasz) a kezdeti pontoknak mintegy 10-20%-nyi részét megengedett megoldásoknak vettük, a többit véletlenszerűen és egymástól függetlenül generáltuk ezek környékén.

A kvadratikus regresszióban szereplő D_k mátrix pozitív definitségét a 6.2.4 alatt leírt módon biztosítottuk – ez még rosszul előállított kezdeti ponthalmazok esetén is működött, csak akkor valamivel több „messze lévő" pontot kellett előállítani (ezek száma a 10-et nem haladta meg). Tehát a kezdeti halmaz jó megadása nem volt annyira fontos, mint kezdetben gondoltuk, mert a ponthalmaz alább leírt adaptív kiszélesítése pótolja a hiányosságokat.

Pontok adaptív megadása

Mint láttuk a 6.2.4 pontban, a kvadratikus regressziós függvény meghatározása lényegében egy $n_q \times n_q$ méretű mátrix invertálását jelenti, ahol $n_q = 1+n+n(n+1)/2$. Az invertálás során néha stabilitási problémák léptek fel, amelyek kezelésére a következő adaptív mechanizmust használtuk. Az előállított $\mathbf{M}$ mátrixot és annak inverzét összeszoroztuk, ha ez a szorzat az egységmátrixtól egy 10^{-3} hibahatárnál jobban tért el, akkor néhány újabb, a pillanatnyi optimális megoldástól egyre távolabb lévő pontot adtunk hozzá a ponthalmazhoz, addig, amíg az invertálás nem lett sikeres.

Ezeket a pontokat a következő módon állítottuk elő. Generáltunk két, az egységgömb felületén egyenletes eloszlású, egymásra ortogonális $\mathbf{u}_1, \mathbf{u}_2$ vektort, és az S_k halmazhoz négy pontot csatoltunk: $\mathbf{x}_{k+1} = \mathbf{x}_k + C_s\mathbf{u}_1, \mathbf{x}_{k+2} = \mathbf{x}_k - C_s\mathbf{u}_1, \mathbf{x}_{k+3} = \mathbf{x}_k + C_s\mathbf{u}_2, \mathbf{x}_{k+4} = \mathbf{x}_k - C_s\mathbf{u}_2$, ahol C_s egy skálázási konstans volt (kezdetben a C_s értékének 0.05-öt választottunk). Ha az így kibővített ponthalmazon meghatározott mátrix, illetőleg az inverze megfelelő volt, akkor folytattuk az SRA algoritmust. Ha az $\mathbf{MM}^{-1}$ szorzat még mindig a hibahatárnál jobban különbözött az egységmátrixtól, akkor a C_s értékét megdupláztuk, és újabb négy pontot adtunk hozzá az S_k halmazhoz. Ezt addig ismételtük, amíg az invertálás nem lett sikeres, a C_s konstans értékét, amely bizonyos értelemben megadta a stabilitáshoz szükséges, „elég széles" ponthalmaz sugarát, későbbi felhasználásra megőriztük. Az SRA algoritmus használatában, az általunk vizsgált példák esetében a ponthalmaz ilyen adaptív szélesítésére általában 2-8 alkalommal volt szükség, az algoritmus első 1000 lépésében – a későbbiekben már nem kellett ezt használni.

Degeneráció elleni védekezés

Az algoritmus végrehajtása során közelítő kvadratikus optimalizálási feladatokat kell megoldani, egy optimális megoldás esetén némely feltételek aktívak, mások pedig nem aktívak lesznek – ezek az aktív feltételek vagy nemnegatívitási, vagy pedig egyenlőtlenséggel megadott feltételek, vagy ezek valamilyen keveréke lesz. Emiatt (hacsak az aktív halmaz az algoritmus előrehaladása során nem változik nagyon gyakran) a kapott közelítő optimális megoldások egy, az aktív feltételek által meghatározott L lineáris sokaságon fekszenek. Mivel ezen lineáris sokaság n_L dimenziós (legfeljebb $(n - m_1)$ dimenziós), így a $q_k(\mathbf{x})$ kvadratikus közelítésünk határértékben degenerálttá válik. Ugyan lehetséges lenne egy, az n_L-dimenziós L sokaságban előállítani egy kvadratikus függvényt, de előre nem tudjuk

eldönteni, hogy adott esetben mi lesz az aktív halmaz az optimum közelében, illetőleg az hogyan változhat meg, ezért a megoldáshoz más utat követtünk.

Az ilyen, degenerációhoz közeli helyzetek elkerülése érdekében nem csak egyetlen $\mathbf{x}_k$ pontot adunk hozzá az S_k halmazhoz az algoritmus 3. lépésében, hanem több pontot. Ezeket a pontokat az utolsó közelítő optimális megoldás kis környezetében vesszük fel. Jelölje a hozzáadott pontok számát n_{add}, ennek értékét a számítógépes futtatások alatt 2 és $2n+2$ közötti értéknek választottuk; az alábbi leírásban $n_{add} = 2n+2$ pontot használtunk, bár a gyakorlatban minden $n_{add} \geq 5$ értékre jó eredményeket kaptunk. Ennek a módosításnak egy további számítástechnikai előnye is volt: csökkent az elvégzendő munka mennyisége. Ugyanis nem minden egyes pont esetén határoztuk meg újra a közelítő kvadratikus függvényt, csak minden n_{add}-edik pont hozzáadása után.

Az n_{add} számú $\{\mathbf{x}_l\}_{l=k+1}^{k+2n+1}$ pótpontot a következő módon konstruáltuk meg. Ahogy szokásos, a k-adik iterációs lépésben megoldott közelítő feladat optimális megoldását $\mathbf{x}_k$-val jelöljük. Legyen adva egy véletlen ortonormalizált $\mathbf{U}$ ponthalmaz, ahol $\mathbf{U} = \{\mathbf{u}_i\}_{i=1}^{n}$, $\mathbf{U}$ egyenletes az ortogonális rendszerek között, $\mathbf{u}_i\mathbf{u}_j = \delta_{ij}$, ahol $\delta_{ii} = 1, \delta_{ij} = 0, i \neq j, i, j = 1, 2, \ldots, n$. Miután az $\mathbf{x}_k$ pontot hozzáadtuk S_k-hoz, csatoljuk még az $\mathbf{x}_{k+j} = \mathbf{x}_k + \gamma_k\mathbf{u}_j$, $\mathbf{x}_{k+n+j} = \mathbf{x}_k - \gamma_k\mathbf{u}_j$ pótpontokat is, ahol $j = 1, \ldots, n$, és $\gamma_k \sim C_s/\sqrt{k}$. Itt γ_k egy olyan változó, amelyet csak lassan engedünk nullához tartani, ennek az értéke kezdetben az előzőleg leírt C_s konstans, vagyis ez az alacsonyabb dimenzióba való eséstől (degenerációtól) óvó állandó. Másrészt az $1/\sqrt{k}$ faktor azt eredményezi, hogy a pótpontok halmaza egyre közelebb kerül a pillanatnyi optimális megoldáshoz – így nem akadályozza a pontsorozat konvergenciáját.

A tapasztalatok szerint a stabilitást viszonylag könnyen el lehetett érni azzal, hogy a generált ortonormált vektorokat két csoportban állítottuk elő. Az első csoport tartalmazta a $f(\mathbf{x}) = \mathbf{c}'\mathbf{x} + q_k(\mathbf{x})$ célfüggvény $\nabla f(\mathbf{x}_k)$ gradiensét és néhány további vektort (itt $n - 1 - n_L$ darab, az L lineáris sokaságra merőleges vektort választottunk), a második csoportban pedig n_L számú, L-beli vektort választottunk (természetesen a két csoportban lévő vektorok egymásra is merőlegesek voltak). Ennek a csoportosításnak az volt az értelme, hogy az első csoportban lévő vektorok a gradiens pontosabb meghatározását segítették elő, míg a második csoportban lévő vektorok segítségével az egymásutáni iterációkban meghatározott közelítő optimum jobban tudott az L sokaságon belül elmozdulni. Az $n_{add} = 5$ számú pótpont esetén a gradienst és negatívját választottuk az első csoportba, a második csoportba pedig a két utolsó közelítő optimális megoldást összekötő egyenes irányvektorát + és - előjellel.

Hozzáadhatunk még egy pótpontot a következők szerint. Az algoritmusban állandóan számoltuk a közelítő minimális függvényértéket, az $f_{appr.min.}$ értékét, amelyet az $\mathbf{x}_{a.min}$ pontban értünk el, ezért ez a pont a rendelkezésünkre áll, és nehézség nélkül csatolható az S_k halmazhoz.

Tehát az $n_{add} = 2n + 2$ számú pótpont a következőképpen áll össze: az $\mathbf{x}_k$ közelítő megoldás, az ortonormált rendszer vektorjai (mind +, mind - előjellel) és a közelítő minimum pontja. Minél nagyobb az n_{add} értéke, annál kevesebb első lépcsős feladatot kell

megoldani, viszont ez lassíthatja a konvergenciát. Ezért érdemes az algoritmus elején csak néhány pontot hozzáadni a ponthalmazhoz, míg a vége felé akár $10n - 100n$ számú pótpont is használható.

A közelítő minimum

Az algoritmus működése során könnyen számon tarthatjuk a közelítő minimum értékét – ezt az algoritmus 3. lépésében, közvetlenül az új függvényérték kiszámítása után tesszük meg. Az algoritmusban előállított közelítő optimális megoldások indexét jelöljük K-val, ugyanis az S_k halmazból csak az első lépcsős feladat determinisztikus feltételeinek eleget tevő $\mathbf{x}_k, k \in K$ pontok függvényértékeit vesszük itt figyelembe. Jelölje az $\mathbf{x}_k, k \in K$ pontban kiszámított zajos q_k függvényérték szórását σ_k, amelyet a tényleges mintából számított empírikus szórás értékével azonosítunk. Az $f_{appr.min.}$ közelítő minimum értéke kezdetben legyen egy nagy szám, és utána ennek értékét minden egyes $k \in K$ index esetén felfrissítjük a

$$f_{appr.min.} = q_k + 3\sigma_k, \text{ ha } q_k + 3\sigma_k \leq f_{appr.min.}$$

egyenlet alapján; ha az egyenletben megadott feltétel nem áll fenn, akkor megnöveljük az $f_{appr.min.}$ értékét (lásd lentebb). Jelölje $\mathbf{x}_{a.min}$ azt a pontot, amely esetén ezt a közelítő minimum értékét találtuk; az $f_{appr.min.}$ a tényleges minimum egy felső korlátjának tekinthető.

Egy kis (0.05) valószínűséggel a kiszámított q_k függvényérték hibája nagyobb lehet $3\sigma_k$-nál is, ha ezt nem vesszük figyelembe, akkor ugyanazt a hibás minimum pontot újra és újra csatolnánk az S_k halmazhoz. Ezért rugalmasabbá tesszük a felfrissítést: ha a jelenlegi közelítő minimumot nem csökkentettük egy $k \in K$ értékre, akkor megnöveltük egy kis értékkel az

$$f_{appr.min.} = f_{appr.min.} + 0.1\sigma_k$$

egyenlet alapján, ahol k az az index, amely esetén az utolsó alkalommal csökkentettük a közelítő minimum értékét. Ezzel a módosítással előbb, vagy utóbb kimozdulunk az aktuális $\mathbf{x}_{a.min}$ pontból (persze még akkor is, ha ez tényleg a minimum pont volt).

Az *SRA* viselkedése és a súlypont

Néhány heurisztikus gondolattal világítjuk meg az algoritmus viselkedését. Ez a SUMT eljárásokra emlékeztet – egy korlátot alkot a $\mathbf{x}_k, k \in K$ pontok számára. Az algoritmus egy $\mathbf{x}_k$ közelítő megoldást számít ki, és aztán újra ellenőrzi, a $q_k \sim \mathcal{Q}(\mathbf{x}_k, \boldsymbol{\xi}_k)$ függvényérték kiszámításával a pont optimalitásának helyességét. Ha q_k kisebb, mint a regressziós függvény értéke ezen a helyen, akkor lefelé módosítja ebben a pontban a $q_k(\mathbf{x})$ függvényt (ha nagyobb, akkor felfelé). Vagyis egy idő után egy többé-kevésbé stabil tálat alakít ki, és a $\mathbf{x}_k, k \in K$ pontok csak ebben tudnak mozogni.

Véletlent használó algoritmusok esetében általában jó ötlet a súlypont vizsgálata. Jelöléseinkkel ez az $\mathbf{M}_1 = (1/\sum \lambda_i) \sum \lambda_i \mathbf{x}_i$ pontot jelenti, ahol az összegezés az $i =$

$0, \ldots, k$ indexekre történik. A közelítés stabilizálása alatt létrehozott „messze" lévő pontok hatása miatt ez a súlypont lényegesen eltérhet az optimális megoldástól, és elég sok időt vehet igénybe ennek a konvergenciája az optimális megoldáshoz. Ezek szerint érdemes egy módosított súlypontot alkotni, amelyben nem szerepeltetjük az algoritmus elején meghatározott, főleg a stabilizálást szolgáló „messze" lévő pontokat, vagyis legyen

$$\mathbf{s} = (1/\sum_{i=1500, i\in K}^{k} \lambda_i) \sum_{i=1500, i\in K}^{k} \lambda_i \mathbf{x}_i.$$

A legtöbb esetben ez az $\mathbf{s}$ pont jobb eredményeket adott (kisebb célfüggvényértéket), mint az utolsó $\mathbf{x}_k$ közelítő megoldás, hiszen az előbbi a tál közepén lassan változtatta a helyét, míg az utóbbi jóval nagyobb ingadozásokkal próbált meg a tál szélének nekimenni.

6.6.2. Súlyozás a regressziós függvény meghatározásában

Megkönnyíti a konvergenciát (főleg az algoritmus első pár száz lépésében), ha súlyokat vezetünk be a minimális normájú közelítés meghatározásában. A súlyok meghatározásában a függvényértékeket és ezek szórásait használtuk. A kvadratikus approximáció $D_k, \mathbf{b}_k, c_k$ paramétereinek meghatározásához eredetileg a

$$\min_{D_k, \mathbf{b}_k, c_k} \sum_{i=0}^{k-1} \left[q_i - (\mathbf{x}_i D_k \mathbf{x}_i + \mathbf{b}_k' \mathbf{x}_i + c_k)\right]^2.$$

minimalizálási feladatot kellett megoldani. Az ebből származó approximáló függvény globálisan jól (vagy rosszul) fogja közelíteni a várható pótlás függvényét, de nekünk nem egy mindenhol jó közelítésre van szükségünk, hanem egy olyan approximációra, amely az optimumhoz közeli pontokban ad jó közelítést. Annak eldöntésére, hogy melyik pont van közel az optimális megoldáshoz, mi a függvényértéket használtuk. Azokban az $\mathbf{x}_i$ pontokban, amelyekben a célfüggvény értéke „nagy" (vagy nagyon eltér a várható optimumtól), megengedhetjük, hogy a fenti, minimalizálandó összegben szereplő $[q_i - q_k(\mathbf{x}_i)]^2$ négyzetes eltérés viszonylag nagy legyen, de azt szeretnénk, hogy a „kis" q_i függvényértékeket adó pontokban ez az eltérés kicsi legyen. Ezt azzal érhetjük el, hogy a fentebbi minimalizálási problémában a „kis" függvényértékű pontokat nagy súllyal szerepeltetjük, míg a „nagy" függvényértékű, számunkra kevésbé fontos pontok esetén fellépő távolságnégyzeteket kis súllyal vesszük figyelembe. Tehát a következő, súlyozott négyzetes hibákat tartalmazó feladatot akarjuk megoldani:

$$\min_{D_k, \mathbf{b}_k, c_k} \sum_{i=0}^{k-1} \lambda_i \left[q_i - (\mathbf{x}_i D_k \mathbf{x}_i + \mathbf{b}_k' \mathbf{x}_i + c_k)\right]^2, \tag{6.33}$$

ahol az $\mathbf{x}_i$ pontban az aktuális zajos függvényérték q_i volt, melyre $\sigma_i^2 = D^2(q_i)$. Annak eldöntésére, hogy melyik függvényérték „nagy", illetőleg „kicsi", az $f_{appr.min.}$ közelítő

minimumértéket használtuk. A (6.33) feladatban szereplő λ_i súlyokat a következő egyenletekből határozzuk meg:

$$\begin{aligned}
\lambda_i &= \lambda_i^{(1)}\lambda_i^{(2)}, \text{ ahol} \\
\lambda_i^{(1)} &= \begin{cases} \frac{1}{[1+(q_i - f_{appr.min.})^2]}, & \text{ha } |q_i - f_{appr.min.}| \geq 5\sigma_i, \\ 1, & \text{egyébként}, \end{cases} \\
\lambda_i^{(2)} &= \begin{cases} \frac{C_\lambda}{\sigma_i^2}, & \text{ha } i \geq 1500, \\ 1, & \text{egyébként}, \end{cases}
\end{aligned} \tag{6.34}$$

ahol C_λ egy konstans volt, amelyet a számításokban használt Θ_7 becslésnek a szokásos N=3, vagy 6 mintaszám esetén elért szórásával tettünk egyenlővé. A súly értékét két tényező szorzataként állítottuk elő. A $\lambda_i^{(1)}$ tényező értéke a minimumtól való távolságot veszi figyelembe és nagyon fontos volt az algoritmus első néhány száz lépésében – ezalatt a második tényezőt egyszerűen $\lambda_i^{(2)} = 1$ értékűnek vettük. A nagyon kis súlyok használata ellen úgy védekeztünk, hogy a súlyokra egy alsó korlátot is használtunk; ha a (6.34)-ből számított első faktorra $\lambda_i^{(1)} \leq 0.001$ következett be, akkor beállítottuk a $\lambda_i^{(1)} = 0.001$ értéket. A $|q_i - f_{appr.min.}| \geq 5\sigma_i$ feltételt azért kellett beiktatni, hogy az optimum egy kis környezetében ne rontsuk el a torzítatlanságot, ezért ezekben az esetekben a $\lambda_i^{(1)} = 1$ értéket használtuk. A második $\lambda_i^{(2)}$ tényező pedig azt vette figyelembe, hogy a kisebb szórású eredményeknek fontosabb szerepet kell játszania a számításokban, mint a nagyobb szórásúaknak.

Míg $\lambda_i^{(1)}$ meghatározása heurisztikus, $\lambda_i^{(2)}$ értéke matematikailag precíz. Ugyanis ha egy adott $\mathbf{x}_i$ pontban négyszer számítjuk ki egy Monte Carlo eljárással a függvényértéket (négyszer ugyanazt az $\mathbf{x}_i$ pontot csatoljuk az S_k halmazhoz), akkor az ekvivalens azzal, hogy abban a pontban csak egyszer számítjuk ki a függvényértéket, de négyszeres mintaszámmal, ezért szerepel itt $\lambda_i^{(2)} = 4$.

A $\lambda_i^{(1)}$ és a $\lambda_i^{(2)}$ faktorokat az algoritmus 3. lépésében határoztuk meg, a függvényértékek kiszámítása után. Ezután az $\mathbf{M}$ kiszámításában fellépő a $m_0 = (1/\sum\lambda_i)\sum\lambda_i q_i$, $m_1 = (1/\sum\lambda_i)\sum\lambda_i q_i \mathbf{x}_i$, $\mathbf{M}_1 = (1/\sum\lambda_i)\sum\lambda_i \mathbf{x}_i$, etc. mennyiségeket frissítettük fel (az összegezés 0-tól k-ig megy) a $\lambda_i q_i$, $\lambda_i \mathbf{x}_i$ értékekkel. Mivel az $f_{appr.min.}$ közelítő minimum értéke (főleg kezdetben) sok bizonytalanságtól függ, ezért néhányszor (a $k = 500, 1500, 3000$ iterációban kapott pontok kiszámítása után) az összes λ_i súlyt újraszámítottuk.

Ez a súlyozási eljárás azt eredményezte, hogy már az első 500-1500 lépésben az algoritmus elért az optimum közelébe – ugyanis az optimum értéke és a közelítő optimum értéke közti különbség már kisebb volt, mint a optimális függvényérték 1 %-a.

Nagyméretű feladataink esetén a λ_i súlyok kiszámításában nem a fentebbi közelítő minimumot használtuk, hanem a Mak, Morton és Wood által leírt módon meghatározott (az elsőlépcsős célfüggvényértékre vonatkozó) alsó korlátot, ami jóval biztonságosabb.

	Mayer	Shapiro-Mello	*SRA*
$f(\mathbf{x})$	15.175542	15.175263	15.175213
σ_f	$\pm$0.000026	$\pm$0.000025	$\pm$0.000025
x_1	0.518506	0.522541	0.523334
x_2	0.0	0.0	0.0
x_3	0.142612	0.142639	0.142644
x_4	0.463797	0.462186	0.462354
x_5	0.054323	0.053281	0.053286
x_6	0.053981	0.057888	0.057398
x_7	0.0	0.0	0.0
x_8	0.0	0.0	0.0
x_9	0.414995	0.419552	0.418981
x_{10}	0.577079	0.576777	0.576717

6.12. táblázat. A közelítő optimális megoldások független komponensek esetén – Mayer, Shapiro, *SRA*.

6.6.3. Számítógépes eredmények

Egy közepes méretű numerikus feladaton szemléltetjük részben az *SRA* algoritmus hatékonyságát, részben a folytonos eloszlású valószínűségi vektorok esetére való alkalmazhatóságát.

A példát eredetileg a Kall és Mayer által kifejlesztett SLP-IOR sztochasztikus programozási programcsomag segítségével állították elő, amelyet a [SH 98] cikkben publikáltak (a publikációban egy nyomdahiba is volt, a megadott -16.36 érték hibás, a helyes érték W(10,13)=-16.33, továbbá az ott megadott szórás értékek valójában a szórásnégyzetek értékei). Mi itt az alapfeladatot és néhány variánsát oldottuk meg – az eredeti feladatban független komponensű normális eloszlású $\boldsymbol{\xi}$ valószínűségi vektor szerepelt, itt bevezettünk korrelációt, illetőleg egy variánsban korrelált normális eloszlást tettünk fel a T mátrix elemeire is, ilyen feladatot tudtunkkal még nem próbáltak megoldani.

A feladatnak $n = n_1 = 10$ első lépcsős döntési változója volt, $n_2 = 15$ második lépcsős változója, az első lépcsőben $m_1 = 5$ egyenlőséges feltétel és $m_2 = 10$ második lépcsős egyenlőséges feltétele volt, így a második lépcsőben szereplő véletlen $\boldsymbol{\xi}$ jobboldali vektor 10 dimenziós volt, amelyről feltették, hogy független komponensű normális eloszlást követ.

A második lépcsős lineáris programozási feladatokat és az első lépcsős közelítő kvadratikus feladatokat a MINOS programcsomag segítségével oldottuk meg. Amikor a második lépcsős feladatokat oldottuk meg, hogy a $Q(\mathbf{x})$ függvényértékekre becslést kapjunk, akkor ugyanazon $\mathbf{x}$, de változó $\boldsymbol{\xi}_i, i = 1, 2, \ldots, N$ jobboldalakra kellett megoldani a lineáris programozási feladatot, ezért a számítási munka csökkentése céljából a HOT opciót használtuk (az utolsó előállított bázisból indult ki az algoritmus). Gyakorlatilag egy adott $\mathbf{x}$

$k=$	$\mathbf{x}_{k,1}$	$f(\mathbf{x}_k)$	Hiba	$\sigma/\sqrt{M}$	M	idő
150	3.975	74.1369	∞	0.01	30	
380	0.59	16.3459	1.17	0.006	70	
1100	0.52	15.2086	0.0334	0.004	200	20 sec
2700	0.496	15.1853	0.0101	0.002	1000	1 min
5900	0.524	15.1764	0.0012	0.0005	13000	10 min
12000	0.524	15.1756	0.0004	0.0002	100000	40 min

6.13. táblázat. Független komponensek, hiba$=|f(\mathbf{x}_k)-f_{appr.min}|$, ahol $f_{appr.min}=15.1752$, $D(\Theta_7)=\sigma_1=0.05$ az $N=1$ mintaszámra, M a k-adik iterációig felhasznált Θ_7 becslések száma.

esetén előállítottuk az összes szükséges $\boldsymbol{\xi}_i$ realizációt és az így kapott feladatokat oldattuk meg a MINOS programcsomaggal. További gyorsulást is elérhettünk volna az azonos optimális bázist adó jobboldalak egy csoportba sorolásával (bunching). Az elért sebesség szempontjából megemlítjük, hogy mintegy 1000 szimplex feladatot oldott meg a gép 1 sec alatt. A pótló függvény kiszámítását legtöbbször a Θ_7 becslés használatával végeztük azonos N mintaszámmal, a mintaszám változtatása és más becslések használata esetén kapott eredmények nem mutattak nagyobb eltérést.

A 6.12 táblázatban adjuk közre a különböző közelítő optimális megoldásokat: az elsőt Mayer adta meg, a második a Shapiro és Homem de Mello által kiszámított, a harmadik oszlopban pedig az *SRA* által meghatározott eredmény van. A második sorban szereplő $f(\mathbf{x})$ optimális függvényértékeket a 6.4 szakaszban megkonstruált becslések segítségével számítottuk ki. Ennél lényegesen pontatlanabbak a Mayer által megadott korlátok $14.9927 < f(\mathbf{x_{M-opt}}) < 15.6822$ (ezt az eredményt 1995-ben kapta), illetőleg a Shapiro és de Mello által megadott $f(\mathbf{x}_{S-M-opt})=15.1866$ érték. A harmadik sorban megadott értékek a második sorban szereplő függvénybecslések szórásai.

A 6.13 táblázatban az *SRA* algoritmus sebességét mutatjuk be. Az első oszlop az iterációs számláló értékét, a második oszlop az $\mathbf{x}_k$ megoldás első komponensét, a harmadik oszlop a célfüggvényértékét, a negyedik az aktuális célfüggvényérték tényleges hibáját tartalmazza. Itt M jelöli a teljes algoritmus k-adik lépéséig felhasznált Θ_7 becslések kiszámításának számát, ahol $N=1$ mintaszámot tételeztünk fel, a becslés szórás ekkor $\sigma_1=D(\Theta_7)|_{N=1}=0.05$ volt (M-nek és σ_1-nek a meghatározásához némi átszámításokat kellett végezni, de ezeket nem ismertetjük). Az itt közölt számok alapján is megerősítést kapott az optimális szórásra vonatkozó sejtésünk: az esetek mintegy 90 %-ában az $|f(\mathbf{x}_k)-f_{appr.min}|$ tényleges hiba kisebb volt, mint $3\sigma_1/\sqrt{M}$. Kihangsúlyozzuk, hogy ezt akkor is elértük, ha a becsléseink pontosságát nem növeltük meg az algoritmus előrehaladása folyamán; a táblázatban adott esetben a legutolsó iterációkban a függvényértékeket 0.01 szórással számítottuk, de a végső eredmény tényleges hibája 0.0004 volt (ami megfelel a sejtett $0.05/\sqrt{M}\sim 0.0003$ szórásnak).

$\varrho =$	0.2	0.4	0.6	0.8	0.9	0.95
M	47000	47000	45000	47000	65000	65000
$f(\mathbf{x})$	15.1412	15.1089	15.0632	15.0229	14.9860	14.9711
σ_f	± 0.0003	± 0.0003	± 0.0004	± 0.0004	± 0.0002	± 0.0002
Eff	10	9	6	7	6	6
x_1	0.51510	0.50859	0.53454	0.50720	0.51886	0.51274
x_2	0.0	0.0	0.0	0.0	0.0	0.0
x_3	0.14259	0.14255	0.14272	0.14254	0.14261	0.14257
x_4	0.46303	0.46344	0.45799	0.46265	0.46213	0.46248
x_5	0.05429	0.05502	0.05044	0.05480	0.05357	0.05425
x_6	0.05620	0.05559	0.06795	0.05772	0.05828	0.05780
x_7	0.0	0.0	0.0	0.0	0.0	0.0
x_8	0.0	0.0	0.0	0.0	0.0	0.0
x_9	0.41758	0.41686	0.43129	0.41936	0.42001	0.41945
x_{10}	0.57733	0.57782	0.57588	0.57792	0.57705	0.57751

6.14. táblázat. Korrelált komponensek, M az összes felhasznált becslés száma, a sejtés alapján a végeredmény szórása $\sigma_f = \sigma_1/\sqrt{M} \sim 0.0002 - 0.0004$.

Ha nagyon nagy M értéket használtunk ($M \sim 10^6 - 10^8$ esetén), akkor a fentebbi, hibára vonatkozó sejtésünk nem teljesül. Ennek az oka a következő lehet: a $Q(\mathbf{x})$ függvény szakaszonként lineáris, valamilyen „csúcsokkal", és az optimumot valamelyik csúcson veszi fel a célfüggvény. Az általunk használt $q_k(\mathbf{x})$ közelítés egy kétszer folytonosan deriválható függvény, tehát nem is várható el, hogy $q_k(\cdot)$ jól közelít egy csak szakaszonként differenciálható függvényt egy csúcs közelében.

Ezek után az alapfeladatba korrelációt vezetünk be. Tegyük fel, hogy a véletlen jobboldali vektor korrelációs mátrixa $C = \{c_{ij}\}, c_{ii} = 1, c_{ij} = \varrho, i \neq j, i, j = 1, 2, \ldots m_2$, ahol a ϱ együtthatónak különböző értékeket adtunk 0.0 és 0.95 között. Ezekre az esetekre vonatkozó eredményeket közlünk a 6.14 táblázatban. Az Eff feliratú sor tartalmazza a Θ_7 becslés hatékonyságát az adott megoldási pontban. A becslések kiszámításához szükséges idők tekintetében megjegyezzük, hogy egy $N = 2500$ mintaszám esetén egy $D^2(\Theta_7)|_{N=2500} = (0.001)^2$ szórásnégyzetű eredményt lehetett kapni 90 sec alatt (egy $D(\Theta_7)|_{N=25} = 0.01$ szórású eredményt 1 sec alatt – ez 0.3% relatív hiba).

Az eredeti feladat egy másik változatát úgy állítottuk elő, hogy csak a véletlen vektor komponenseinek szórását változtattuk meg, a komponensek függetlenségét és a többi adatot változatlanul hagytuk. Az eredeti feladatban a $\boldsymbol{\xi}$ véletlen vektor minden komponensére a $D(\boldsymbol{\xi}) = 0.1E(\boldsymbol{\xi})$ egyenlőség adta meg a szórást. A 6.15 táblázatban közölt optimális megoldások azokra az esetekre vonatkoznak, amikor a szórásokat rendre $D(\boldsymbol{\xi}) =$

σ	$0.03E(\boldsymbol{\xi})$	$0.15E(\boldsymbol{\xi})$	$0.20E(\boldsymbol{\xi})$	$0.30E(\boldsymbol{\xi})$
$f(\mathbf{x})$	14.61774	15.6224	16.0756	16.9896
σ_f	± 0.00004	± 0.0008	± 0.0008	± 0.0015
x_1	0.509151	0.520185	0.486159	0.481257
x_2	0.0	0.0	0.0	0.0
x_3	0.142549	0.142623	0.142395	0.142362
x_4	0.466537	0.464472	0.469037	0.471498
x_5	0.056309	0.054471	0.059360	0.060843
x_6	0.047506	0.052115	0.042589	0.036535
x_7	0.0	0.0	0.0	0.0
x_8	0.0	0.0	0.0	0.0
x_9	0.407438	0.412816	0.401700	0.394635
x_{10}	0.577778	0.576952	0.579497	0.579864

6.15. táblázat. Különböző szórású független komponensek, $M \sim 100000$ becslés kiszámítása.

$0.03E(\boldsymbol{\xi}), D(\boldsymbol{\xi}) = 0.15E(\boldsymbol{\xi}), D(\boldsymbol{\xi}) = 0.20E(\boldsymbol{\xi}), D(\boldsymbol{\xi}) = 0.30E(\boldsymbol{\xi})$ értékűekre változtattuk meg.

Az utolsó változatra vonatkozó eredményeket a 6.16 táblázatban közöljük. Ebben az esetben az eredeti feladatnak egy olyan variánsát állítottuk elő, amelyben a $\boldsymbol{\xi}$ vektor mellett a második lépcsős T mátrix elemei is valószínűségi változók. Jelölje a T mátrix (i, j)-edik nemzérus elemét t_{ij}. Ezen nem-zérus értékű elemek helyett tekintsük a τ_{ij} valószínűségi változókat, amelyeknek együttes eloszlása legyen normális. Legyen a várható értékük az előző t_{ij} érték, szórásnégyzetük pedig $\sigma_{ij}^2 = (t_{ij}/10)^2$. Így a feladatunkban 31 valószínűségi változó van, a $(\boldsymbol{\xi}, T)$ valószínűségi vektorváltozó egy 31-dimenziós normális eloszlást követ, mivel a T mátrixnak 21 nemzérus eleme volt.

Az ilyen módon előállított kétlépcsős feladatra vonatkozó eredményeinket közöljük az utolsó 6.16 táblázatban. Az első oszlopban arra az esetre adjuk meg az optimális megoldást, amikor a $(\boldsymbol{\xi}, T)$ valószínűségi vektorváltozó komponensei függetlenek, a második és a harmadik oszlopban azoknak az eseteknek a közelítő optimális megoldását adtuk meg, amikor a valószínűségi vektor komponensei korreláltak voltak $\varrho = 0.5$, illetőleg $\varrho = 0.9$ korrelációval. Végül az utolsó oszlopban megadott eredményeket úgy kaptuk, hogy a valószínűségi vektor komponensei között $\varrho = 0.9$ korrelációt tettünk fel, továbbá az összes komponens szórását megnöveltük a várható értékük 0.3-szeresére. A korrelációs mátrix struktúrája a fentebbi esetekben ugyanolyan volt, mint az előző esetekben, vagyis a főátlóban csupa 1-es állt, a főátlón kívüli elemek mindegyikének értéke ϱ volt.

	független	$\varrho = 0.5$	$\varrho = 0.9$	$0.3E(\boldsymbol{\xi}, T)$
$f(\mathbf{x})$	15.523327	15.301289	15.015361	16.313862
σ_f	± 0.007	± 0.005	± 0.003	± 0.006
x_1	0.504974	0.514245	0.538690	0.441198
x_2	0.0	0.0	0.0	0.0
x_3	0.142521	0.142583	0.142747	0.142094
x_4	0.465697	0.468022	0.459209	0.485912
x_5	0.056304	0.056513	0.050611	0.070502
x_6	0.049973	0.043306	0.064510	0.001858
x_7	0.0	0.0	0.0	0.0
x_8	0.0	0.0	0.0	0.0
x_9	0.410317	0.402537	0.427280	0.354170
x_{10}	0.578090	0.577397	0.575569	0.582859

6.16. táblázat. Véletlen T mátrix a második lépcsős feladatban; független komponensek, korrelált komponensek $\varrho = 0.5$, $\varrho = 0.9$ korreláció esetén, végül $\varrho = 0.9$ megnövelt szórással.

6.7. Nagyméretű feladatok

Ebben a szakaszban nagyméretű kétlépcsős feladatok futásával kapcsolatban szerzett számítógépes tapasztalatainkat írjuk le. Mivel az előző szakaszokban szereplő kisebb méretű példákon az algoritmus viselkedését részletesen elemeztük, ezért ebben a szakaszban csak azt mutatjuk meg, hogy az algoritmus használható nagyméretű feladatok megoldására is.

Tekintsük a kétlépcsős feladatot a szokásos formában:

$$\begin{aligned} \min\ \mathbf{c}'\mathbf{x} &+ Q(\mathbf{x}), \\ \text{f.h. } A\mathbf{x} &= \mathbf{b}, \\ \mathbf{x} &\geq 0, \\ Q(\mathbf{x}) = E(\mathcal{Q}(\mathbf{x}, \boldsymbol{\xi})) &= E(\min_{\mathbf{y}} \mathbf{q}'\mathbf{y} | W\mathbf{y} = \boldsymbol{\xi} - T\mathbf{x}, \mathbf{y} \geq 0), \end{aligned} \tag{6.35}$$

ahol az A, T, W mátrixok rendre $m_1 \times n_1, m_2 \times n_1, m_2 \times n_2$ méretűek, a $\mathbf{b}$ és $\boldsymbol{\xi}$ vektorok m_1 illetőleg m_2 dimenziósok, a többi vektor pedig megfelelő dimenziós. A feladatokban csak a második lépcsős feladat jobboldala, vagyis a $\boldsymbol{\xi}$ volt véletlen az ezen szakaszban vizsgált példák esetén. Ezekről feltettük, hogy független komponensű normális eloszlást követnek. A vizsgált feladatok mérete $n_1 = 20 - 100$, illetőleg $m_2 = 20 - 120$ között változott.

Az ebben a szakaszban leírt számítógépes futásokat egy 1,9 GHz AMD processzorral és 1 Gbyte memóriával rendelkező személyi számítógépen hajtottuk végre, Windows XP rendszert használva (a Windows 95, Windows 98, Windows 98SE és hasonló rendszerek nem tudják kezelni a 1/2 Gbyte-nál nagyobb memóriát, illetőleg a virtuális memória kezelése nem működik megfelelően).

6.7.1. A számítógépes módosítások

A 6.6 szakaszban leírtakhoz képest néhány módosítást hajtottunk végre a számítógépes programon azzal a céllal, hogy nagyméretű feladatokat is meg tudjunk oldani elfogadható idő alatt. Az alábbiakban ezeket a módosításokat ismertetjük.

Véletlen példák előállítása

Megfelelő mennyiségű és minőségű feladat előállításához egy véletlen példákat generáló szubrutinrendszert írtunk. A programban a Kall és Mayer által készített SLP-IOR sztochasztikus programozási programcsomagban is használt, véletlen kétlépcsős feladatokat generáló algoritmust valósítottuk meg. Ennek segítségével sokféle feladatot állítottunk elő, hiszen elég a program elején a kiindulásként használt véletlenszámgenerátort eggyel tovább léptetni és teljesen másik feladatot kapunk.

Ez a véletlen példákat generáló szubrutinrendszer teljes pótlású feladatokat generál. A numerikus példa előállításában tetszőlegesen lehet választani a.) a feladat részeinek dimenzióit, b.) megadhatók a mátrixok sűrűségei és c.) megadható egy intervallum, amelyben az együtthatóknak benne kell lennie. A rendszer részletes leírása megtalálható Mayer könyvében ([May 98] 100-103. oldalak).

A lefuttatott példák közül százhúsz darab teljes leírását, valamint az *SRA* algoritmus által meghatározott közelítő megoldását közzétettük a `www.math.bme.hu/~deak` címen. Az ott megadott fájlok tartalmazzák a numerikus feladat teljes leírását, továbbá az *SRA* algoritmus működése közben előállított részeredményeket és az általunk kapott utolsó közelítő megoldást. A fájlok tartalmának részletesebb leírását a `readme.text` fájl tartalmazza. Egy numerikus példát és megoldását a függelékben adtunk meg.

Sherman-Morrison

Mivel a memóriaigény és a futási idők nagy mértékben függenek az $\mathbf{M}$ mátrix invertálásától, ezért az iterációk nagy részében az inverz kiszámítását a Sherman-Morrison felújítással végeztük el (lásd a 6.2.3 részt). A numerikus stabilitást azzal őriztük meg, hogy 500-5000 újabb pont hozzáadása után teljes inverziót végeztünk.

A várható értéket is tartalmazó célfüggvényt közelítő kvadratikus függvényt a 6.6 szakaszban leírtak szerint kerestük, vagyis a

$$\min_{D_N,\mathbf{b}_N,c_N} \sum_{i=1}^{N} \lambda_i \left[q_i - (\mathbf{x}_i D_N \mathbf{x}_i + \mathbf{b}'_N \mathbf{x}_i + c_N)\right]^2, \tag{6.36}$$

súlyozott eltérések összegét minimalizáltuk, ezért a megfelelő felújítás kifejezése az alábbiak szerint számolható ki. Jelölje most $\boldsymbol{\xi}_i = \boldsymbol{\xi}_i(\mathbf{x}_i)$ a 6.2.3 pontban leírt módon az $\mathbf{x}_i$ vektorból kialakított $n_Q = n_1(n_1+1)+n_1+1$ dimenziós vektort és jelöljük ezek diadikus szorzatai közül az első N súlyozott összegét $\overline{\mathbf{M}}$, vagyis

$$\overline{\mathbf{M}} = \mathbf{M} \sum_{i=1}^{N} \lambda_i = \sum_{i=1}^{N} \lambda_i \boldsymbol{\xi}_i \boldsymbol{\xi}'_i.$$

Vezessük be az $L = \sum_{i=1}^{N} \lambda_i$, $\mathbf{M}^{-1} = \left[\frac{1}{L}\overline{\mathbf{M}}\right]^{-1}$ és $\lambda = \lambda_{N+1}$ jelöléseket is. Ekkor a következő

$$\mathbf{M}^{(N+1)} = \frac{1}{L+\lambda}\left[\sum_{i=1}^{N} \lambda_i \boldsymbol{\xi}_i \boldsymbol{\xi}'_i + \lambda \boldsymbol{\xi}_{N+1} \boldsymbol{\xi}'_{N+1}\right] = \frac{L}{L+\lambda}\mathbf{M} + \frac{\lambda}{L+\lambda}\boldsymbol{\xi}_{N+1}\boldsymbol{\xi}'_{N+1} \tag{6.37}$$

mátrixot és annak $\left[\mathbf{M}^{(N+1)}\right]^{-1}$ inverzét kell meghatározni további számításainkhoz, ezt pedig a következő egyenlőség adja meg

$$\left[\frac{1}{L+\lambda}\overline{\mathbf{M}}^{(N+1)}\right]^{-1} = \frac{L+\lambda}{L}\left\{\mathbf{M}^{-1} - \frac{\mathbf{M}^{-1}\boldsymbol{\xi}_{N+1}\boldsymbol{\xi}_{N+1}{}'\mathbf{M}^{-1}}{\frac{L}{\lambda} + \boldsymbol{\xi}_{N+1}{}'\mathbf{M}^{-1}\boldsymbol{\xi}_{N+1}}\right\}.$$

Konfidencia intervallum

Beépítettük a programba a Mak, Morton és Wood [MMW 99] szerzőtársak által megadott eljárást, amely segítségével a célfüggvényre vonatkozó alsó korlátot lehet kiszámítani, illetőleg az optimalitási hézagra konstruált 0.95 megbízhatóságú konfidencia intervallumot lehet előállítani – ezzel tetszőleges $\mathbf{x}_k$ pont esetén ellenőrizhető, hogy mennyire jó ez a közelítés.

Először a Mak-Morton-Wood által leírt eljárás lényegét vázoljuk. Könnyen lehet alsó korlátot meghatározni az optimális függvényértékre, ha a második lépcső jobboldalán szereplő $\boldsymbol{\xi}$ vektort néhány (számításainkban ez $n_{bl} = 10-100$ között volt) véletlenszerűen generált realizációjából álló diszkrét eloszlással helyettesítjük. Legyenek a $\boldsymbol{\xi}$ realizációi $\boldsymbol{\xi}^{(1)}, \ldots, \boldsymbol{\xi}^{(n_{bl})}$, ekkor feladatunk a következő

$$
\begin{array}{rcccl}
\min \mathbf{c}'\mathbf{x} & + & \frac{1}{n_{bl}} \sum_{i=1}^{n_{bl}} \mathbf{q}'\mathbf{y}^{(i)} & & \\
A\mathbf{x} & & & = & \mathbf{b} \\
T\mathbf{x}+ & W\mathbf{y}^{(1)} & & = & \boldsymbol{\xi}^{(1)} \\
\vdots & & & \vdots & \\
T\mathbf{x}+ & \cdots & +W\mathbf{y}^{(n_{bl})} & = & \boldsymbol{\xi}^{(n_{bl})} \\
\mathbf{x} \geq 0, & \mathbf{y}^{(1)} \geq 0, & \ldots, & \mathbf{y}^{(n_{bl})} \geq 0. &
\end{array}
\tag{6.38}
$$

Az így kapott feladat egy nagyobb, $[m_1 + n_{bl}m_2] \times [n_1 + n_{bl}n_2]$ méretű lineáris programozási feladat, n_{bl} itt az elsőlépcsős feladatot kiegészítő W mátrixok (blokkok) száma a keletkező nagyméretű lineáris programozási feladatban. Jelölje O az eredeti feladat optimális célfüggvényértékét, $O^*_{n_{bl}}$ az n_{bl} darab véletlen realizációval előállított közelítő feladat optimális célfüggvényértékét. Ekkor az $E[O^*_{n_{bl}}] \leq E[O^*_{n_{bl}+1}] \leq O$ egyenlőtlenség fennáll, tehát nagyobb méretű mintával (várható értékben) pontosabb alsó becslést kapunk kapunk az optimális célfüggvényértékre.

Ilyen becslést egy példára $m_{batch} = 10 - 100$ alkalommal hajtunk végre, amiből az alsó korlát becslésének szórását is meg lehet határozni, illetőleg a becslések átlagának szórását is megbecsülhetjük. Az így kapott m_{batch} számú becslés átlagát Q_{MMW}-vel jelöljük, ez az alsó korlát egy becslése. Megjegyezzük, hogy mivel az ezen m_{batch} számú becslés kiszámítása folyamán kapott $(\mathbf{x}, \mathbf{y}^{(1)}, \ldots, \mathbf{y}^{(n_{bl})})$ megoldások első része (a közelítő feladat megoldásának első n_1 komponense, amely az első lépcsőre vonatkozik) az első lépcső feladatának megengedett megoldásai, ezért ezeket a pontokat is felhasználtuk a további számításokban (ezeket a pontokat hozzáadtuk a kezdeti ponthalmazhoz).

A bizonytalanabb (6.2 szakaszban leírt) ideiglenes minimum helyett ezt a Q_{MMW} alsó korlátot használtuk a λ_i súlyok kiszámításánál; az adott pontban számított függvényérték és az alsó korlát távolságából számítottuk a függvényérték és a közelítő kvadratikus alak értéke közti eltérés λ_i súlyát.

Az SRA algoritmus utolsóként kapott $\mathbf{x}_k$ közelítő megoldása jóságának megitélésére pedig megadható egy konfidencia intervallum, amely az $|f(\mathbf{x}_k) - O|$ optimalitási hézag nagyságát mutatja 95%-os megbízhatósággal. Megjegyezzük, hogy a konfidencia intervallum meghatározásánál az $f(\mathbf{x}_k) = \mathbf{c}'\mathbf{x}_k + Q(\mathbf{x}_k)$ függvényérték kiszámításához ugyanazokat a véletlen jobboldali vektorokat használtuk, mint a Q_{MMW} alsó korlát kiszámításánál, hiszen így a becslés szórásának csökkentését értük el (common random numbers vagy másképpen control variates technique, lásd [MMW 99]). Ezzel a CRN technikával azt is elértük, hogy az optimalizálási hézagra kapott becslés mindig pozitív lett. További szóráscsökkentést értünk el azzal, hogy a 6.5 szakaszban leírt szimmetrizálási eljárás alapján a generált vektorokat és azok -1-szeresét is felhasználtuk a generált mintában, tehát a MMW alsó korlát kiszámításában nem $\boldsymbol{\xi}^{(1)}, \ldots, \boldsymbol{\xi}^{(n_{bl})}$ független mintákat vettük, hanem a $\boldsymbol{\xi}^{(1)}, -\boldsymbol{\xi}^{(1)}, \ldots, \boldsymbol{\xi}^{(n_{bl})/2}, -\boldsymbol{\xi}^{(n_{bl})/2}$ vektorokat (lásd a Monte Carlo integrálásnál leírt szimmetrizálást).

Az $\mathbf{x}_k$ megoldás jóságát azzal jellemeztük, hogy ez a konfidenciaintervallum a $f(\mathbf{x}_k)$ célfüggvényérték hány százaléka, a numerikus eredmények között ezért ez a százalék mutatja az elért eredmény jóságát.

Megállási feltételek

Az *SRA* algoritmus előző, vázlatos leírásaiban az ott szereplő „elég jó" feltétel teljesülését egy több összetevőből álló megállási kritériummal helyettesítettük. Mivel véletlen algoritmusok esetén a hibák megzavarhatják a konvergenciát, mi megköveteltük, hogy feltételeleink teljesüljenek az utolsó három iterációban számított értékekre. Három részből áll optimalitási kritériumunk, az első a célfüggvény értékeinek stabilitását biztosítja, a másik két feltétel pedig egy bizonyos stabilitást igényel az egymásutáni közelítő megoldásoktól.

a.) Jelölje a σ_{MMW} a Q_{MMW} az alsó korlátra vonatkozó becslés szórását, $\sigma_i = D[f(\mathbf{x}_i, \boldsymbol{\xi})] = D[\mathbf{c}'\mathbf{x}_i + \mathcal{Q}(\mathbf{x}_i, \boldsymbol{\xi})]$ a függvényértékek szórását. Ekkor megkívántuk, hogy a függvényértékek stabilan az alsó becslés közelében legyenek, vagyis hogy a

$$|f(\mathbf{x}_i) - Q_{MMW}|^+ \leq 0.005|f(\mathbf{x}_k) + Q_{MMW}| + \frac{3.0}{2}(\sigma_{MMW} + \sigma_i), i = k-3, k-2, k-1, k$$

egyenlőtlenségek teljesüljenek. A statisztikai becslések miatti bizonytalanság természete miatt nem a függvényértékek különbségét korlátozzuk, hanem a biztonságosabb pozitív részt vettük.

b.) Az egymásutáni közelítő $\mathbf{x}_k$ optimális megoldások kevéssé változzanak egymáshoz viszonyítva, vagyis megköveteltük a

$$\frac{1}{3}\sum_{i=k-2}^{k} |\mathbf{x}_i - \mathbf{x}_{i-1}| \leq 0.01|\mathbf{x}_k|$$

egyenlőtlenség teljesülését, ahol a pontok eltérését az euklideszi normában vettük.

c.) Végül kikötöttük még, hogy az utolsó négy megoldás esetén ugyanaz legyen a közelítő optimális megoldásokra az aktív (nemnegativitási) feltételek halmaza.

Az így megadott összetett megállási kritérium jól működött, mert a megfelelő konfidenciaintervallum általában kicsi lett (1-2% körüli). Megállási kritériumként természetesen használhattuk volna a MMW eljárásban leírt konfidencia-intervallum hosszát is, például megkövetelve azt, hogy ez a konfidenciaintervallum a függvényérték egy százalékánál kisebb legyen. Azért nem ezt az utat választottuk, mert ez számítástechnikailag időigényesebb lett volna, hiszen minden iteráció után ki kellett volna számítani az optimalitási hézagra vonatkozó becslést.

A numerikus példák nehézségi foka

Szükség volt arra, hogy valamilyen egyszerű mérőszámot vezessünk be az egyes feladatok nehézségének durva megbecslésére. Ezért a továbbiakban használni fogjuk a feladat

	$n_1 = 20$ $k_{in} = 354$		$n_1 = 40$ $k_{in} = 1024$		$n_1 = 60$ $k_{in} = 2094$	
$m_2 = 20$	átlag	min , max	átlag	min , max	átlag	min , max
`idő (sec)`	20 sec	[8,35]	213 sec	[118,284]	2412 sec	[2219,3610]
`k pont (db)`	859	[380,1781]	2560	[1485,3435]	5506	[5252,5988]
`nehézség`	0.49	[0.0,1.31]	0.83	[0.13,1.88]	0.49	[0.18,0.86]
`konfidencia`	1.2%	[0.0,3.2]	3.6%	[1.0,8.4]	5.7%	[1.5,11.0]
$m_2 = 40$	átlag	min , max	átlag	min , max	átlag	min , max
`idő (sec)`	234 sec	[100,381]	651 sec	[386,1174]	2636 sec	[1458,3467]
`k pont (db)`	686	[427,854]	2515	[1541,5948]	4832	[4125,6004]
`nehézség`	0.36	[0.1,0.86]	0.37	[0.10,1.23]	0.41	[0.19,1.37]
`konfidencia`	0.9%	[0.2,2.2]	1.4%	[0.2,2.8]	4.3%	[0.3,15.0]
$m_2 = 60$	átlag	min , max	átlag	min , max	átlag	min , max
`idő (sec)`	947 sec	[476,1503]	2923 sec	[1448,8905]	5090 sec	[2799,6936]
k`pont (db)`	532	[431,684]	2362	[1248,10198]	3974	[2622,5572]
`nehézség`	0.12	[0.04,0.28]	0.16	[0.05,0.23]	0.22	[0.07,0.63]
`konfidencia`	0.2%	[0.0,0.7]	1.2%	[0.3,6.9]	1.0%	[0.2,3.2]

6.17. táblázat. Eredmények átlaga 10-10 darab, 20-60 dimenziós kétlépcsős feladatra: teljes futási idő, az előállított pontok száma, a feladat nehézségi foka és az optimalitási hézag konfidenciaintervallumának nagysága.

nehézségi fokát, amelyet a

$$d = \frac{3D\left[\mathbf{c}'\mathbf{x} + \mathcal{Q}(\mathbf{x}, \boldsymbol{\xi})\right]}{\mathbf{c}'\mathbf{x} + Q(\mathbf{x})} \tag{6.39}$$

aránnyal adunk meg. Itt a számlálóban álló $D\left[\mathbf{c}'\mathbf{x} + \mathcal{Q}(\mathbf{x}, \boldsymbol{\xi})\right]$ kifejezés a legegyszerűbb, durva mintavétellel történő Monte Carlo kiszámítás esetén fellépő szórás nagysága, a nevező pedig a teljes elsőlépcsős célfüggvény értéke. Ez a mennyiség nagyjából azt mutatja meg, hogy az általunk használt Monte Carlo integrálási eljárás hibája a függvényértékhez hogyan aránylik. Természetesen ez a mennyiség függ a $\mathbf{x}$ értékétől, az alábbi numerikus eredményeknél ezért a nehézségi fokot a következő módon határoztuk meg: kiszámítottuk a hányadost a Mak-Morton-Wood eljárásban kapott megengedett pontok mindegyikében, ennek átlagát vettük, valamint kiszámítottuk d-t a legutolsó közelítésben és ennek a két értéknek az átlagát vettük. Különböző paraméterekkel futtatva ugyanazt a numerikus példát, az így kiszámított nehézségi fok viszonylag stabilnak mutatkozott és a szükséges futási időkkel arányosnak.

6.7.2. Számítógépes eredmények

Az alábbiakban 124 darab, véletlenszerűen generált numerikus példa eredményeit többnyire csak összefoglalóan mutatjuk be. A példák teljes leírása a részletes futási eredményekkel együtt a megadott web címen találhatók tíz könyvtárban, az L1-L9 könyvtárakban 12-12 feladat van, az L10 könyvtárban 16 darab. Az L1-L9 könyvtárakban vannak a kisebb feladatok, az utolsó könyvtár tartalmazza a legnagyobb méretűeket. Feladatainkban a kezdeti pontok k_{in} számát úgy határoztuk meg, hogy teljesüljön a

$$k_{in} \geq n_1(n_1+1)/2 + n_1 + 1$$

egyenlőtlenség. A minimálisan szükséges $n_1(n_1+1)/2+n_1+1$ véletlenszerűen előállított kezdeti ponthoz hozzáadtuk még a Mak-Morton-Wood eljárás folyamán a Q_{MMW} alsó becslés kiszámítása során kapott m_{batch} számú megengedett pontot (lásd a 6.7.1 pontban írtakat), továbbá az n darab (pozitív) egységvektor egy állandóval szorzott értékét is.

Egy példára a továbbiakban $n_1 X m_2.y$ kifejezéssel hivatkozunk, ahol n_1, m_2 a megfelelő dimenziók, y pedig az adott dimenziópárra generált példák közül az y-odik változat (a web-en megadott numerikus adatok ilyen nevű fájlokban találhatók). A feladatok nem feltüntetett dimenzióit most és a további feladatok esetén is az $m_1 = n_1/2, n_2 = 3m_2/2$ összefüggések határozzák meg. A véletlen jobboldalak szórását $D(\boldsymbol{\xi}) = 0.3E(\boldsymbol{\xi})$ értékűnek vettük, hogy elég nehezen megoldható feladatokat kapjunk (Mayer példáiban a szórás általában kisebb).

A nagyméretű feladatok megoldására vonatkozó első 6.17 táblázatunk 90 példa eredményének átlagát tartalmazza; a példák a 20X20.1–20X20.10, ..., 60X60.1–60X60.10 jelű fájlokban találhatók és az $n_1 = 20, 40, 60$, $m_2 = 20, 40, 60$ dimenziók összes lehetséges párjára előállított tíz-tíz darab változata futásának átlagát adjuk meg. Az ebben a táblázatban összefoglalt példák esetén a generálási és futtatási paramétereket nem változtattuk meg, az A, T, W mátrixok sűrűsége rendre 40%, 30% és 10% volt, a Monte Carlo integrálást a Θ_3 becsléssel végeztük, a mintaszám $KS = 15$ volt. Az itt közölt átlagok stabil mérőszámai a futásoknak, más paraméterértékekkel futtatva az adott dimenziós példákat az értékek kevéssé térnek el a táblázatban megadottaktól.

Minden dimenziópárban a következő sorok találhatók: az `idő` sora a teljes algoritmus futási idejét jelenti, a pontszám az előállított k darab pont száma (vagyis a k_{in} kezdeti pontok száma és az SRA által előállított pontok együttes száma), a nehézségi fok a példákra a (6.39) egyenlőséggel definiált érték volt, míg az optimalitási hézagra adott 95%-os megbízhatóságú konfidenciaintervallumot a teljes elsőlépcsős célfüggvényérték százalékában adtuk meg (nyilvánvalóan kisebb százalék pontosabb eredményt jelent). Minden dimenziópárban és adatsorban három adatot adtunk meg: az átlagot, valamint a legkisebb és legnagyobb értéket (például a 20X20 típusú feladatok teljes futási idejének adatai szerint a tíz példa futási idejének átlaga 20 sec, a legkisebb és legnagyobb futási idő 8 sec és 35 sec volt).

Egy adott példa nehézségi foka és az utolsó közelítő megoldásra kapott konfidencia intervallum nagysága általában pozitív korrelációt mutat, a nehezebb feladat megoldása

40X20										
nehézségi fok	0.13	0.25	0.25	0.36	0.58	0.69	0.98	1.30	1.87	1.88
konfidencia %	2.6	1.6	1.0	6.0	1.5	1.6	2.5	2.6	1.4	8.4
60X60										
nehézségi fok	0.07	0.11	0.15	0.16	0.18	0.19	0.21	0.22	0.32	0.63
konfidencia %	0.2	0.2	0.5	0.5	0.6	0.5	0.7	2.3	3.2	1.1

6.18. táblázat. A nehézségi fok és a konfidenciaintervallum nagysága.

során általában csak nagyobb konfidenciaintervallumot képes meghatározni az algoritmus. Jól látható ez a 40X20 és a 40X40 példák átlagain a 6.17 táblázatban, ahol az átlagos nehézség 0.83, illetőleg 0.38 volt, az átlagos konfidenciaintervalum pedig 3.6%, illetőleg 1.4% volt – annak ellenére, hogy a véletlen vektor dimenziója a kétszeresére nőtt. A nehézségi fok és a konfidencia intervallum nagysága közti összefüggés bemutatására közöljük a 40X20 és a 60X60 típusú feladatok nehézségi fokra és konfidenciaintervallumra vonatkozó eredményeit a 6.18 táblázatban, a példákon kapott egyes értékeket növekvő nehézségi fok szerint felsorolva (a 60X60-as példák jól mutatják ezt a korrelációt, a 40X20-as példák kevésbé).

A 6.19 táblázatban négy példára (a 20X20.1, 20X60.1, 60X20.1, 60X60.1 példákra) a teljes algoritmus egyes részeinek futási ideit, illetőleg egyéb adatait adjuk meg. A példák száma után feltüntettük a nehézségi fok és a százalékban megadott konfidenciaintervallum arányát is. Az adatok közül először az SRA algoritmus futási idejét adjuk meg, majd az utolsó közelítés k sorszámát, a kezdeti pontok k_{in} számát, ebben a k_{in} pontban a Q függvény Monte Carlo kiszámításának idejét és végül egy $n_Q \times n_Q$ dimenziós $\mathbf{M}$ mátrix invertálásának idejét (ahol $n_Q = n_1(n_1+1)/2 + n_1 + 1)$). Ezek után a Mak-Morton-Wood (MMW) féle becslések kiszámításának idejét, a megoldandó (legnagyobb méretű) lineáris programozási feladat sorainak és oszlopainak számát, valamint az utolsó közelítés esetén a „pontos” függvényérték kiszámításának idejét. A táblázat utolsó sorában a teljes algoritmus futási idejét adjuk meg. Itt jegyezzük meg, hogy a MMW becslés meghatározása viszonylag sok időt vett igénybe, mert a standard MINOS programot használtuk a nagyméretű, duál-dekomponálható LP megoldására – ezen természetesen lehet gyorsítani például a Ruszczynski-féle regularizált dekompozíció alkalmazásával. Amint az várható volt, az n_1 értékétől erősen függ az SRA algoritmus futási ideje, míg a MMW becslések kiszámítása az m_2 értékétől.

A következő 6.20 táblázatban azt mutatjuk meg, hogy bár az utolsó közelítés időnként nem nagyon pontos (a konfidenciaintervallum 1%-nál nagyobb volt az esetek mintegy felében, a 6.17 táblázatban összefoglalt kilencven példa közül háromszor volt 10%-nál nagyobb), de a paraméterek változtatásával (pontosabb Monte Carlo integrálással – a mintaszám növelésével, vagy módszer változtatásával, illetőleg nagyobb n_{bl} értékkel –

Példa	20X20.1	20X60.1	60X20.1	60X60.1
nehézség/konfid.	0.28/0.9%	0.05/0.1%	0.43/2.8%	0.11/0.2%
SRA ideje	7 sec	43 sec	2232 sec	911 sec
– k pontok száma	720	470	5476	2622
– k_{in} =	354	354	2094	2094
– k_{in} fv.érték	1 sec	20 sec	5 sec	92 sec
– egy invertálás	1 sec	1 sec	280 sec	278 sec
MMW ideje	4 sec	759 sec	18 sec	2350 sec
– LP mérete	611×920	1811×2720	631×960	1831×2760
– „pontos” fv.érték	3 sec	150 sec	20 sec	186 sec
Teljes idő	15 sec	995 sec	2255 sec	3645 sec

6.19. táblázat. Négy példa futásának részletes adatai; itt például a 20X60.1 feladat dimenziói $n_1 = 20, m_1 = 10, m_2 = 60, n_2 = 90$.

Példa	k =	KM/KS	$f(\mathbf{x}_k)$”pontos”	Q_{MMW}	n_{bl}/m_{batch}	konfid.	idő
60X20.71	5660	3 / 15	52.29±0.4	47.84±0.8	20 / 30	9.9%	3610 sec
60X20.72	8561	3 / 50	51.72±0.1	49.80±0.5	50 / 50	4.8%	7410 sec
60X20.73	5969	7 / 4	51.10±0.1	50.29±0.5	50 / 50	3.6%	4336 sec
60X20.74	6773	7 / 8	50.64±0.1	50.32±0.3	100 / 50	2.5%	5643 sec

6.20. táblázat. A megoldás pontosságának növelése – egy példa négy futása, különböző Monte Carlo integrálási paraméterekkel.

Példa	$k=$	nehézs.	konfid.	$f(\mathbf{x}_k)$"pontos"	Q_{MMW}	n_{bl}/m_{batch}	idő
80X80.13	5830	0.09	0.2%	6656.0±0.8	6641.0±4.3	20 / 50	7920 sec
80X80.14	4653	0.10	0.2%	649.6±0.1	646.4±0.8	20 / 50	5774 sec
80X80.15	10086	0.21	3.7%	700.4±0.1	387.4±2.3	20 / 30	18219 sec
80X80.16	10201	0.33	3.0%	397.4±0.1	389.8±1.9	20 / 50	24672 sec

6.21. táblázat. Az algoritmus futási jellemzőinek változása különböző sűrűségű W mátrixok esetén – első példa 3%, második 4%, harmadik és negyedik példa 5%.

pontosabb alsó korlát kiszámításával) az eredmény pontosabbá tehető. Ezt a 60X20.7 feladat négy, változtatott paraméter beállítással kapott, a 60X20.71, 60X20.72, 60X20.73, 60X20.74 jelű (L7 könyvtárban található) futások összefoglalásával szemléltetjük. A feladat közepesen nehéznek mutatkozott, nehézségi foka 0.8-1.0 közötti értékeket vett fel a különböző futásokban. Az eredmények szerint a pontosság növelhető megfelelő futtatási paraméterek beállításával. A táblázat oszlopai rendre a következő mennyiségeket tartalmazzák: az utolsó közelítés sorszáma (az előállított pontok száma), a Monte Carlo integrálásban használt becslés (KM) illetőleg a mintaszám (KS), az utolsó pontban az elsőlépcsős célfüggvény „pontos" értéke a szórással, a Q_{MMW} alsó korlát becslése és a szórása, a Q_{MMW} kiszámítása során használt n_{bl}, illetőleg m_{batch} paraméterek értékei, az optimalitási hézagra kapott konfidenciaintervallum és végül a teljes algoritmus futási ideje. Jól látható, hogy az utolsó két sorban a Θ_7 becslés alkalmazása hatékonyabb, mint a Θ_3 becslés, valamint az, hogy nagyobb n_{bl}, m_{batch} értékekre szorosabb alsó korlátot kapunk.

A web-en közreadott L10 könyvtárban találhatók a legnagyobb méretű numerikus feladatok, amelyeket lefuttattunk. Itt 16 feladat található, melyek méretei 20X120, 100X20, 80X80, és 100X120 volt, mindegyik típusból négy darab.

A 80X80 típusú feladatokban a W mátrix sűrűségét változtattuk, a 80X80.13 feladatban ez a sűrűség 3%, a 80X80.14 feladatban 4%, a 80X80.15 és 80X80.16 feladatokban 5% volt, a kapott eredményeket a 6.21 táblázatban mutatjuk be. Jól látható, hogy a W mátrix sűrűségének növelése részben az MMW alsó becslés kiszámítása, részben a $\mathcal{Q}$ függvényérték (Monte Carlo becslés) kiszámítási idejének emelkedését eredményezi, amint az várható volt.

Az L10 könyvtárban található feladatok közül háromnak (20X120.1, 100X20.1, 100X120A) a részletesebb eredményeit és paraméterértékeit mutatjuk be az 6.22 táblázatban. Az itt bemutatott 100X120A jelű feladat számítástechnikai szempontból a legrosszabb viselkedésű volt az ilyen méretű feladatok közül, míg a Függelékben közölt 100X120B feladat a legjobb viselkedésű volt ezek között. Megjegyezzük, hogy a 100X120A feladat futási eredményeiben a Q_{MMW} alsó korlát (és ezzel 100 megengedett megoldás) előállításához volt szükség 2257 sec időre, a további 920 megengedett megoldás előállításához (amelyeket a k_{in} kezdeti ponthalmazhoz csatoltunk) további 8940 sec időre volt szükség – hasonló, az

Példa	20X120.1	100X20.1	100X120A
dimenziók	$n_1 = 20, m_2 = 120$	$n_1 = 100, m_2 = 20$	$n_1 = 100, m_2 = 120$
nehézség/konfidencia	0.24 / 1.0%	0.43 / 0.9%	0.65 / 3.6%
SRA ideje	1107 sec	32111 sec	51518 sec
– k pontok száma	1299	12767	17591
– $k_{in} =$	1024	6354	6354
– k_{in} fv.érték	174 sec	27 sec	363 sec
– egy invertálás	1 sec	4755 sec	4750 sec
– $f(\mathbf{x}_k)$ „pontos”	719.9±0.3	96.66±0.1	1004.1±0.3
MMW ideje	615 sec	110 sec	2257 sec
– alsó korlát	713.5±3.6	95.84±0.3	967.4±6.7
– LP mérete	2411 × 3620	1051 × 1600	2931×4420
– „pontos” érték idő	360 sec	3 sec	595 sec
Teljes idő	4502 sec	32414 sec	63311 sec

6.22. táblázat. Három nagyméretű példa eredménye.

m_2 és az n_{bl} értékeitől függő idők adhatók meg a többi feladatnál is.

Az utolsó, 6.23 táblázatban 4-4 darab, 20X120, 100X20, illetőleg 100X120 típusú feladat futási eredményeinek átlagát adjuk meg. A teljes algoritmus futási idejét itt percekben adtuk meg.

Röviden összegezzük tapasztalatainkat. A bevezető részekben írtaknak megfelelően az előállított kvadratikus közelítés nem pozitív definit, és rendre rossz helyen keresi a minimumot. Ezeken a helyeken további pontot és függvényértéket ad hozzá az algoritmus az S_k ponthalmazhoz és ezzel kijavítja a defektust a függvényben (mintegy önjavító algoritmusként). Általában $k_{in} - 2k_{in}$ újabb pont generálása után az algoritmus megtalálja az optimális megoldás(ok) környezetét. Mivel ekkor már a kvadratikus közelítés is stabil, ezért ezután már csak ebben a környezetben finomítja a kvadratikus függvényt (lásd a 6.20 táblázatban feltüntetett eredményeket, vagy az előző, 6.6 szakaszban írottakat).

A futási időkre vonatkozó eredményekből látható, hogy míg az m_2 dimenzió (a véletlen vektor dimenziója) viszonylag könnyen növelhetőnek tűnik, az első lépcsős döntési változó n_1 dimenziójának növelése nehéznek mutatkozik az SRA algoritmus használata esetén.

A legnagyobb, 100X120 típusú feladataink összehasonlítható méretűek a Sen és társai által megoldott feladattal [SDC 93]. Az általuk megadott telekommunikációs probléma 86 véletlen változót tartalmaz, ezek diszkrét eloszlásúak voltak és összesen 10^{70} különböző változatot (scenario) vehetnek fel, az A mátrix 1×86-os, a második lépcsős W mátrix

	20X120		100X20		100X120	
	átlag	min , max	átlag	min , max	átlag	min , max
idő	52 min	[37,75]	631 min	[540,794]	785 min	[525,1055]
k pont	1158	[956,1320]	14256	[12767,16653]	15269	[12005,17591]
nehézs.	0.16	[0.06,0.24]	0.75	[0.25,1.95]	0.46	[0.14,0.65]
konfid.	0.4%	[0.0,1.0]	1.8%	[0.3,4.7]	3.2%	[0.6,5.1]

6.23. táblázat. Nagyméretű feladatok: 4-4 darab adott dimenziós példa eredményének átlaga, $n_1 = 20 - 100, m_2 = 20 - 120$ dimenzióban, az időeredmények percben vannak megadva.

170×706-os méretű, 1.85%-os sűrűségű volt. A mi 100X120 típusú feladatainkban 120 dimenziós független komponensű, normális eloszlású valószínűségi vektorváltozó szerepelt, az A mátrix 50×100-as, a W mátrix 120×180-as méretű és 1.95% – 2.0% sűrűségű volt. A Mak, Morton és Wood által, a Ruszczynski-Swietanowski kóddal megoldott Senféle probléma megoldására 8%-os konfidenciaintervallumot kaptak mintegy 2600 min idő alatt, míg a mi nagyméretű feladataink esetén átlagosan 3.2%-os konfidenciaintervallumot kaptunk (a legrosszabb esetben 5.1% volt), átlagosan 785 min alatt. Az összehasonlítás nem ad okot komoly következtetések levonására, mert Mak és társai cikkében nincsen megadva a számítógép és jellemzői. Mindezek ellenére elmondható, hogy nagyméretű feladatok megoldhatók az SRA algoritmus használatával.

6.8. Összefoglalás

Az előző fejezet végén tettünk már néhány megjegyzést az SRA eljárás és a sztochasztikus approximáció kapcsolatáról. Az SRA algoritmus hasonlít a statisztikai és mérnöki számításokban gyakran használt válasz-felületi módszerre (Response Surface Methodology – RSM); rávilágítunk az RSM és a szukcesszív regressziós approximációk közötti különbözőségre.

Az RSM eljárást G.E.P. Box és munkatársai fejlesztették ki 1951-ben, újabb alkalmazásait és formáit lásd például Draper és Smith [DS 66] valamint Khuri és Cornell [KC 87] könyvét, valamint a [De 02] kötetét. A Válasz-felületi eljárás egy sztochasztikus optimalizálási alkalmazását láthatjuk Marti munkáiban, ahol egy sztochasztikus approximációt használó hibrid algoritmust közöl [Mar 88], [Mar 92]: az eljárás alapjában véve sztochasztikus approximáció, de a lépések egyik részében egy zajos (véges differenciákat használó) gradiensbecslést alkalmaz, a lépések egy másik részében néhány közeli pontban függvényértéket, majd ezeken átmenő regressziós görbét számít ki és ennek az analitikus gradiensét használja közelítő gradiensként.

Az eljárást Box és Draper könyve ([BD 87] 182-188. oldalakon) alapján írjuk le, ahogy azt egy zajos függvény feltétel nélküli maximalizálására adták meg (lásd [BD 87] 183.o.).

Tekintsünk egy megengedett megoldást, és vegyünk fel ennek a környékén néhány pontot és ezekben a függvényértékeket, ezek alkotják az első ponthalmazt. Ezek segítségével határozzunk meg egy kvadratikus alakú regressziót. Tekintsük ennek a gradiensét és vegyünk fel további pontokat a gradiens mentén (ezek alkotják a második ponthalmazt). Ezt a második halmazt addig bővítjük, amíg a függvényértékek nem kezdenek csökkenni – a minimum közelítő értékének vagy ezt a pontot, vagy a második ponthalmazon kifeszített egydimenziós kvadratikus közelítés által adott maximumpontot vesszük. Az eljárást vagy befejezzük ezzel, vagy (elhagyva az összes eddigi pontot) újra felveszünk néhány pontot, azokra újabb regressziót határozunk meg, stb. (A válasz-felületi algoritmusok kutatásaiban elég nagy részt foglal el egy „hatásos" ponthalmaz megkonstruálásának munkája: hogyan lehet egy adott k egész esetén a legjobb közelítést adó k pontot meghatározni.) Természetesen azonos pontosságot (szórást) feltételezve minden függvénykiszámításnál, és ugyanannyi pontot használva a hiba minden egyes új közelítés esetén ugyanakkora lesz. Látszik, hogy az SRA és az RSM közötti lényeges különbség abban áll, hogy az SRA megtartja az összes eddigi pontot, ami segítségével a tapasztalatok szerint egy fokozatosan csökkenő hibájú közelítést kapunk.

Ez az eljárás három lényeges pontban különbözik az SRA algoritmustól. Először is csak a második (az utolsó) ponthalmazt használja a regressziós függvény meghatározására. Másodszor is, nem látható annak felismerése, hogy az összes előző pont használata konvergenciára vezethet. Harmadszor, nincsen kisérlet arra, hogy valamilyen matematikai bizonyítást konstruáljanak annak megmutatására, hogy a második halmaz segítségével meghatározott pont valóban jobb megoldást adna, mint a kiindulásként használt pont. Bár időnként említés szintjén szerepelnek megjegyzések arról, hogy az algoritmusban régebbi pontokat is lehet használni, a szerző legjobb tudomása szerint senki sem említette, vagy vizsgálta azt, hogy az összes előzőleg meghatározott pont használata esetleg konvergens sorozatot eredményezhetne. Az SRA-t az RSM-hez hasonlítva nevezhetjük egy aszimptotikus válasz-felületi eljárásnak visszacsatolással, hiszen az új közelítésből adódó pontot mindig hozzáadjuk a ponthalmazhoz, így a ponthalmazban szereplő pontok száma a végtelenhez tart.

Végezetül néhány megjegyzést teszünk az SRA algoritmus sztochasztikus programozási alkalmazása során elért számítógépes eredményekkel kapcsolatban.

1. A kétlépcsős feladatokban lényegében ugyanazokat az optimális megoldásokat, lényegében azonos idő alatt lehet meghatározni az SRA eljárással, mint az eddigi legjobb módszerekkel, de a célfüggvényértékekben elért pontosság nagyságrendekkel jobb, mint a Mayer és Shapiro által elért eredmények (Shapiro [SH 98] egyébként nem publikált futási időt, személyes találkozásunkkor ismételt érdeklődésemre is csak annyit mondott, hogy nagyon gyors az algoritmus). Az SRA algoritmus még az általunk használt kissé lassú, 133 MHz-es számítógéppel is elő tudott állítani 1-3 perc alatt egy olyan közelítő optimális megoldást, amelyen a függvényérték tényleges hibája az $f(\mathbf{x})$ értékének legfeljebb 0.1%-a.

2. Az SRA algoritmus a kétlépcsős feladattípus esetén a szokásos diszkretizálási megközelítést használó megoldó algoritmusoknak egy használható alternatívája. Az al-

goritmusunk sokkal kevésbé érzékeny a valószínűségi változók számára, mint az ismert eljárások – lásd a fentebbi, 31 korrelált valószínűségi változót tartalmazó feladatot – ami nagyobb dimenziós feladatoknál fontos lehet. (Ugyanis a diszkretizálásnál szükséges munka exponenciálisan nő a dimenziószám növelésével, míg a Monte Carlo módszerek ennél sokkal lassabban növő munkaigényűek.) Ráadásul képes korrelált eseteket is kezelni, lényegében ugyanolyan hatékonyan, mint a független komponensű eloszlásokat – a változók korreláltsága esetén a diszkretizálás a független esetnél jóval nehezebb numerikus feladatot eredményez.

3. Az *SRA* algoritmussal a valószínűségi korlátos feladatokat is meg lehet oldani, illetőleg valószínűségi korlátot és pótló függvényt együttesen tartalmazó feladatokat (lásd Prékopa vegyes modelljét).

4. A sztochasztikus programozásban gyakran használt lineáris sztochasztikus programozási feladatok helyett az *SRA* algoritmus képes megoldani olyan feladatokat is, amelyekben kvadratikus függvényeket használunk – akár a célfüggvényben, akár a feltételi függvények között.

5. Fontos tulajdonsága az *SRA* algoritmusnak, hogy a számítások során előállított összes pontot megtartjuk. Ez pontosan az ellentéte annak, amit az általános vélekedés szerint minden operációkutatásban dolgozó csinálna. Hiszen ha közel vagyunk az optimumhoz, akkor elvetjük az összes távoli pontot, hogy ne zavarják a konvergenciát. De ezzel éppen a stabilitást (a megfelelő pozitív definit mátrixszal rendelkező kvadratikus approximáció kifeszítését) teszik tönkre.

6. Megjegyezzük, hogy az *SRA* algoritmus különbözik a sztochasztikus approximációtól, illetőleg a Benveniste és társai [BMP 90] által tárgyalt adaptív algoritmusoktól, mert itt a lépéshossz egy valószínűségi változó, nem pedig előre megadott sorozat.

7. Ha zajos függvényeink vannak az optimalizálási eljárás során, akkor az általánosan követett eljárás nagy vonalakban a következőképpen írható le. Az éppen aktuális x_k pontban meghatározzuk a $f(x_k) + \varepsilon_k$ függvényértéket, ahol a $\sigma = D(\varepsilon_k)$ szórást a hiba mértékének tekintjük; nyilván σ értéke a számítási munka mennyiségétől függ. Ezután a következő x_{k+1} pontba jutunk el, ahol a mintavételt újra elvégezzük, ugyanakkora σ szórású eredményt kapunk (az előző függvényértéket pedig elhagyjuk). Ha az x^* optimális pont közelébe jutunk, akkor szeretnénk a pontosságot növelni, így megnöveljük a mintaszámot. Az *SRA* eljárásban erre nincs szükségünk, mert egyszerűen azáltal, hogy megtartjuk az összes előző függvényértéket a pontosság növekedni fog.

8. A számítógépes tapasztalatok szerint a végeredmény pontossága tetszőlegesen növelhető, a sejtésünk szerint a legjobb elvárható szóráscsökkenés fennáll. Tehát ha a zajos függvényt σ szórással M-szer számítjuk ki az eljárás során, akkor a végeredmény pontossága nem σ, hanem asszimptotikusan $\sigma/\sqrt{M}$ nagyságrendű.

9. Az *SRA* algoritmus segítségével az ismert legnagyobb méretű kétlépcsős feladattal összehasonlítható méretű feladatokat is meg lehet oldani.

Függelék

PROBLEM 100X120B

THE DIMENSIONS OF THE TWO-STAGE PROBLEM:
FIRST STAGE HAS 50 ROWS AND 100 VARIABLES
SECOND STAGE HAS 120 ROWS AND 180 VARIABLES

334 NONZEROS OF MATRIX A (ROW AND COLUMN INDICES)
ITS DIMENSIONS: 50 100, WITH DENSITY 0.06

1. (14, 1) -2.80| 2. (21, 1) -1.40| 3. (40, 1) -6.20| 4. (10, 2) 0.90| 5. (38, 2) 8.40| 6. (48, 2) -8.20| 7. (9, 3) 2.40| 8. (17, 3) -4.80| 9. (19, 3) -5.40| 10. (7, 4) 9.40| 11. (22, 5) -3.30| 12. (45, 6) 5.20| 13. (6, 7) -8.30| 14. (15, 7) 0.90| 15. (17, 7) 7.90| 16. (26, 7) 7.20| 17. (27, 7) 1.20| 18. (34, 7) 1.30| 19. (35, 7) -6.40| 20. (38, 7) -1.00| 21. (49, 7) 2.70| 22. (4, 8) 4.00| 23. (5, 8) -5.70| 24. (9, 8) 8.60| 25. (10, 8) -4.50| 26. (20, 8) -4.60| 27. (23, 8) 3.90| 28. (24, 8) 8.40| 29. (40, 8) -3.80| 30. (10, 9) -6.00| 31. (21, 10) 1.60| 32. (30, 10) 4.10| 33. (21, 11) 1.40| 34. (9, 12) 3.00| 35. (14, 12) 1.40| 36. (18, 12) -6.90| 37. (21, 12) -9.80| 38. (26, 12) -0.50| 39. (31, 12) -0.80| 40. (35, 12) 6.20| 41. (37, 12) 8.50| 42. (39, 12) -2.40| 43. (35, 13) -7.30| 44. (20, 14) 0.90| 45. (29, 14) 1.80| 46. (30, 14) 5.40| 47. (7, 15) 4.10| 48. (2, 16) 1.60| 49. (15, 16) -3.00| 50. (16, 16) -0.40| 51. (38, 16) 4.20| 52. (21, 17) 1.50| 53. (32, 17) 7.50| 54. (36, 17) 2.90| 55. (48, 17) -2.80| 56. (25, 18) -4.50| 57. (2, 19) 2.50| 58. (25, 20) 4.40| 59. (35, 21) -5.00| 60. (7, 22) -9.50| 61. (11, 22) 3.60| 62. (17, 22) -1.30| 63. (18, 22) 7.50| 64. (22, 22) 9.30| 65. (23, 22) 5.10| 66. (25, 22) -8.40| 67. (40, 22) -2.40| 68. (7, 23) -4.80| 69. (41, 24) 0.90| 70. (3, 25) 2.90| 71. (7, 25) 2.60| 72. (45, 25) -3.10| 73. (47, 25) 6.30| 74. (38, 26) 4.90| 75. (4, 27) 6.40| 76. (11, 27) -3.60| 77. (14, 27) -9.20| 78. (39, 27) -0.70| 79. (21, 28) 2.90| 80. (1, 29) 8.80| 81. (30, 29) 5.60| 82. (35, 29) -2.30| 83. (39, 29) -7.90| 84. (41, 29) -8.00| 85. (44, 29) -3.40| 86. (34, 30) -6.90| 87. (43, 30) -0.70| 88. (44, 31) -8.60| 89. (50, 32) -3.30| 90. (2, 33) -3.20| 91. (7, 34) -9.50| 92. (23, 34) 0.70| 93. (6, 35) -2.20| 94. (9, 35) 4.40| 95. (15, 35) -0.50| 96. (16, 35) -2.50| 97. (25, 35) -7.00| 98. (28, 35) 3.80| 99. (36, 35) -7.50| 100. (41, 35) 9.70| 101. (47, 35) -1.10| 102. (21, 36) 2.50| 103. (29, 36) -9.50| 104. (11, 37) 3.70| 105. (12, 37) 4.10| 106. (13, 37) 9.50| 107. (24, 37) 8.20| 108. (29, 37) 5.50| 109. (30, 37) -3.60| 110. (31, 37) 6.80| 111. (41, 37) 9.10| 112. (46, 37) 6.20| 113. (2, 38) -4.50| 114. (3, 38) -2.50| 115. (10, 38) 2.50| 116. (20, 38) 5.40| 117. (39, 38) -4.70| 118. (43, 38) 3.40| 119. (44, 38) 8.50| 120. (47, 38) -3.30| 121. (49, 38) -9.60| 122. (37, 39) -9.50| 123. (28, 40) 4.10| 124. (38, 41) -2.50| 125. (7, 42) -6.50| 126. (8, 42) 7.00| 127. (14, 42) -6.00| 128. (23, 42) -8.10| 129. (30, 42) -1.40| 130. (37, 42) 3.30| 131. (39, 42) 3.20| 132. (45, 42) 0.30| 133. (48,

42) -8.90| 134. (3, 43) 6.30| 135. (11, 44) -0.10| 136. (12, 44) 6.40| 137. (11, 45) -5.30| 138. (29, 45) -7.00| 139. (36, 45) 5.10| 140. (38, 46) -4.50| 141. (4, 47) -0.70| 142. (17, 47) -6.00| 143. (21, 47) -9.10| 144. (28, 47) -8.30| 145. (33, 47) 7.80| 146. (1, 48) 7.10| 147. (16, 48) 1.70| 148. (41, 48) 0.40| 149. (45, 48) 0.70| 150. (37, 49) -8.20| 151. (5, 50) 5.90| 152. (31, 50) 9.20| 153. (46, 50) 1.80| 154. (7, 51) -0.40| 155. (30, 51) 3.00| 156. (49, 51) -8.70| 157. (30, 52) 4.30| 158. (8, 53) -5.20| 159. (21, 53) 6.60| 160. (24, 53) -2.10| 161. (48, 53) 4.90| 162. (5, 54) 2.20| 163. (21, 55) -4.30| 164. (24, 55) -8.80| 165. (48, 55) -1.90| 166. (50, 55) -6.50| 167. (9, 56) -9.50| 168. (39, 56) 1.50| 169. (42, 56) -5.30| 170. (2, 57) 6.10| 171. (3, 57) 8.40| 172. (9, 57) 9.70| 173. (15, 57) 3.80| 174. (16, 57) -8.60| 175. (18, 57) 6.00| 176. (25, 57) 3.20| 177. (26, 57) 1.70| 178. (40, 57) 1.20| 179. (19, 58) 8.60| 180. (2, 59) 3.30| 181. (29, 59) -8.00| 182. (33, 59) 8.90| 183. (44, 59) -9.30| 184. (49, 59) -2.30| 185. (1, 60) -4.50| 186. (3, 60) -6.50| 187. (6, 60) 5.70| 188. (28, 60) 3.30| 189. (29, 60) -8.50| 190. (32, 60) 9.20| 191. (33, 60) 0.80| 192. (36, 60) 8.00| 193. (49, 60) 8.30| 194. (8, 61) -8.10| 195. (21, 61) -5.40| 196. (24, 61) -0.50| 197. (26, 61) -0.50| 198. (37, 61) -2.20| 199. (39, 61) -2.00| 200. (40, 61) 2.50| 201. (41, 61) -5.60| 202. (49, 61) -2.40| 203. (16, 62) 0.20| 204. (17, 62) -1.90| 205. (21, 62) 7.10| 206. (29, 62) 1.70| 207. (10, 63) -0.40| 208. (2, 64) 7.50| 209. (16, 64) 5.90| 210. (18, 64) 4.10| 211. (21, 64) -3.30| 212. (29, 64) -8.60| 213. (4, 65) 6.80| 214. (10, 65) 8.50| 215. (27, 66) 5.70| 216. (40, 67) 9.10| 217. (50, 67) -8.20| 218. (8, 68) -1.90| 219. (34, 69) 7.00| 220. (21, 70) 7.40| 221. (28, 70) -3.50| 222. (38, 70) -8.40| 223. (9, 71) -6.70| 224. (10, 71) -6.90| 225. (30, 71) -1.60| 226. (38, 72) 3.20| 227. (44, 72) -8.10| 228. (47, 72) -3.60| 229. (48, 73) -4.00| 230. (20, 74) 1.30| 231. (21, 74) 5.20| 232. (27, 74) -3.30| 233. (32, 74) 4.60| 234. (50, 74) -0.70| 235. (16, 75) -4.50| 236. (43, 76) -6.20| 237. (13, 77) 5.30| 238. (20, 77) -8.00| 239. (23, 77) -7.10| 240. (28, 77) 6.60| 241. (38, 77) 7.60| 242. (42, 77) 2.80| 243. (44, 77) 7.60| 244. (46, 77) -1.90| 245. (49, 77) 5.50| 246. (13, 78) -3.30| 247. (17, 79) -8.50| 248. (37, 79) 9.40| 249. (46, 79) -9.50| 250. (36, 80) -7.30| 251. (9, 81) 3.00| 252. (11, 81) 3.70| 253. (15, 81) 8.30| 254. (18, 81) 2.30| 255. (21, 81) 5.10| 256. (41, 81) 4.50| 257. (48, 81) 1.70| 258. (49, 81) -7.70| 259. (50, 81) -4.10| 260. (1, 82) 8.50| 261. (13, 82) -4.20| 262. (26, 82) -7.00| 263. (47, 82) -7.40| 264. (4, 83) -3.70| 265. (6, 83) -2.00| 266. (7, 83) 0.90| 267. (10, 83) -9.40| 268. (11, 83) 5.40| 269. (13, 83) 1.90| 270. (15, 83) -0.80| 271. (28, 83) -4.30| 272. (39, 83) -6.30| 273. (8, 84) 1.30| 274. (9, 84) -5.30| 275. (12, 84) -3.90| 276. (13, 84) -9.60| 277. (21, 84) 0.50| 278. (29, 84) -7.40| 279. (39, 84) -3.70| 280. (45, 84) 5.30| 281. (49, 84) -0.70| 282. (5, 85) -2.40| 283. (18, 85) -8.30| 284. (22, 85) 7.90| 285. (30, 85) -1.60| 286. (9, 86) 4.40| 287. (18, 86) 6.70| 288. (20, 86) 4.30| 289. (21, 86) 8.70| 290. (26, 86) -4.20| 291. (29, 86) 4.10| 292. (31, 86) 7.20| 293. (34, 86) -4.60| 294. (44, 86) 1.10| 295. (16, 87) -1.40| 296. (15, 88) -9.10| 297. (22, 88) -1.70| 298. (23, 88) 6.90| 299. (29, 88) 5.00| 300. (34, 88) -3.30| 301. (40, 88) -7.70| 302. (36, 89) -5.90| 303. (2, 90) -0.80| 304. (6, 90) 9.20| 305. (7, 90) -1.50| 306. (10, 90) -7.30| 307. (12, 90) 8.10| 308. (13, 90) 1.10| 309. (26, 90) -3.00| 310. (39, 90) 4.40| 311. (45, 90) 2.50| 312. (14, 91) -2.90| 313. (15, 91) -4.50| 314. (27, 91) -9.50| 315. (28, 91) -3.90| 316. (29, 91) 6.60| 317. (30, 91) -3.20| 318. (35, 91) -9.80| 319. (36, 91) 9.30| 320. (43, 91) 3.10| 321. (19, 92) 3.40| 322. (10, 93) -5.90| 323. (6, 94) 4.10| 324. (17, 94) 1.80| 325. (30, 94) -0.10| 326. (47, 94) -6.30| 327. (47, 95) -2.30| 328. (18, 96) -9.70| 329. (42, 96) -8.20| 330. (26, 97) 9.70| 331. (32, 98) -7.80| 332. (39, 99) 2.30| 333. (7,100) 9.20| 334. (19,100) 2.80|

LINEAR PART OF FIRST STAGE OBJECTIVE FUNCTION:
-0.20 0.10 -0.20 0.20 0.00 0.00 -0.10 0.00 -0.10 0.10 0.00 0.30 -0.30 0.20 0.10 0.00 0.10 -0.10 0.00 0.10 -0.20 -0.20 -0.10 0.00 0.10 0.00 0.00 0.00 0.00 -0.20 -0.20 0.00 0.00 -0.20 0.10 0.00 1.30 0.30 -0.30 0.10 0.00 -0.20 0.20 0.20 0.00 0.00 -0.20 0.10 -0.30 0.20 0.00 0.10 0.00 0.00 -0.40 -0.10 0.80 0.10 0.00 0.10 -0.20 0.00 0.00 0.20 0.40 0.00 0.10 0.00 0.20 -0.20 -0.20 -0.20 0.00 0.10 0.00 -0.10 0.40 -0.10 -0.10 0.00 0.10 0.00 -0.30 -0.50 -0.30 0.40 0.00 -0.20 0.00 0.40 -0.50 0.00 -0.10 0.10 0.00 -0.60 0.10 0.00 0.00 0.20

RIGHT HAND SIDE VECTOR OF FIRST STAGE (FOR = CONSTRAINTS)
-0.450 -0.200 -0.900 0.970 0.000 1.490 -1.790 0.310 -0.550 -2.250 0.380 0.830 0.100 -1.810 0.380 0.000 -0.600 -0.900 0.000 0.640 2.880 0.790 -0.350 1.450 0.000 -0.720 -0.710 -1.240 -0.770 -0.840 2.320 1.350 1.750 -0.450 -0.980 1.290 0.330 -0.770 0.000 0.530 1.360 -1.350 0.580 -0.780 1.330 0.800 -0.920 -0.510 -2.070 -1.300

A FEASIBLE SOLUTION OF THE FIRST STAGE:
0.0 0.0 0.0 0.0 0.0 0.1 0.0 0.1 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.1 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.1 0.1 0.0 0.1 0.0 0.0 0.0 0.1 0.0 0.0 0.1 0.1 0.0 0.0 0.1 0.1 0.0 0.0 0.0 0.0 0.1 0.0 0.0 0.1 0.1 0.0 0.1 0.1 0.0 0.1 0.0 0.0 0.1 0.1 0.0 0.0 0.1 0.0 0.0 0.1 0.1 0.0 0.1 0.0 0.1 0.1 0.1 0.1 0.0 0.1 0.0 0.0 0.0 0.0 0.0 0.0 0.1 0.1 0.0 0.0 0.1 0.1 0.1 0.0 0.0 0.0 0.1 0.1 0.0 0.1 0.0 0.1 0.1 0.0 0.1 0.0 0.0

481 NONZEROS OF MATRIX T (ROW AND COLUMN INDICES)
ITS DIMENSIONS: 120 100, WITH DENSITY 0.04
1. (16, 1) 1.10| 2. (61, 1) 1.10| 3. (23, 2) 6.50| 4. (37, 3) 2.40| 5. (61, 3) 3.80| 6. (73, 3) -4.90| 7. (79, 3) 3.20| 8. (86, 3) 9.10| 9. (92, 3) -7.10| 10. (103, 3) 0.10| 11. (68, 4) -8.60| 12. (73, 5) -1.20| 13. (83, 5) 8.80| 14. (109, 5) -6.80| 15. (36, 6) 7.30| 16. (61, 6) -6.40| 17. (76, 6) 5.00| 18. (6, 7) -9.30| 19. (10, 7) -4.50| 20. (11, 7) 2.80| 21. (12, 7) -8.50| 22. (22, 7) -4.10| 23. (29, 7) 8.90| 24. (34, 7) 8.20| 25. (36, 7) 8.10| 26. (63, 7) 9.10| 27. (94, 7) -5.00| 28. (97, 7) -3.60| 29. (105, 7) -4.70| 30. (107, 7) 8.90| 31. (55, 8) -9.70| 32. (89, 8) 7.00| 33. (93, 8) 4.40| 34. (115, 8) -1.00| 35. (37, 9) 8.50| 36. (47, 9) 2.00| 37. (53, 9) 5.70| 38. (59, 9) 2.50| 39. (76, 9) 5.20| 40. (16, 10) -4.90| 41. (40, 10) 0.10| 42. (46, 10) -0.30| 43. (51, 10) 3.60| 44. (71, 10) 6.10| 45. (110, 10) 9.60| 46. (5, 11) 4.50| 47. (30, 11) 0.40| 48. (55, 11) -4.40| 49. (60, 11) 1.50| 50. (70, 11) 1.50| 51. (81, 11) 5.40| 52. (44, 12) 6.10| 53. (67, 12) -9.00| 54. (84, 12) 7.30| 55. (85, 12) 1.40| 56. (21, 13) 3.60| 57. (40, 13) -9.70| 58. (84, 13) 5.00| 59. (85, 13) 2.90| 60. (120, 13) 0.50| 61. (1, 14) 9.60| 62. (12, 14) -9.40| 63. (24, 14) -3.10| 64. (29, 14) -6.70| 65. (32, 14) 2.20| 66. (64, 14) 6.80| 67. (116, 14) 0.40| 68. (15, 15) -0.40| 69. (35, 15) -1.90| 70. (52, 15) 4.90| 71. (85, 15) -7.00| 72. (108, 16) 5.80| 73. (65, 17) -0.40| 74. (70, 17) -6.30| 75. (74, 17) 6.80| 76. (86, 17) 5.90| 77. (93, 17) 5.20| 78. (108, 17) -5.60| 79. (32, 18) 5.40| 80. (35, 18) 5.40| 81. (7, 19) -7.10| 82. (11, 19) -6.40| 83. (18, 19) 6.50| 84. (23, 19) -3.00| 85. (43, 19) -6.40| 86. (76, 19) -1.60| 87. (80, 19) -4.60| 88. (2, 20) -6.90| 89. (27, 20) -4.10| 90. (31, 20) -6.10| 91. (47, 20) -5.30| 92. (50, 20) 5.50| 93. (81, 20) 5.90| 94. (2, 21) -9.70| 95. (26, 21) -6.00| 96. (37, 21) -3.50| 97. (38, 21) -0.80| 98. (53, 21) -8.10| 99. (72, 21) 2.50| 100. (79, 21) -1.70| 101. (5, 22) -2.00| 102. (12, 22) 4.20| 103. (42, 22) -9.50| 104. (49, 22) 9.30| 105. (97, 22) -1.90| 106. (111, 22) -7.80| 107. (115, 22) 0.40| 108. (45, 23) 7.50| 109. (51, 23) 5.30| 110. (76, 23) 2.60| 111. (7, 24) 8.40| 112. (115, 25) -0.90| 113. (31, 26) 1.10| 114. (44, 26) -1.30| 115. (80, 26) 4.20| 116. (88, 26) 7.00| 117. (102, 26) -6.10| 118. (116, 26)

-0.80| 119. (8, 27) -3.40| 120. (51, 27) -6.70| 121. (53, 27) 3.10| 122. (83, 27) 5.50| 123. (94, 27) 5.10| 124. (109, 27) 4.40| 125. (116, 27) -0.40| 126. (17, 28) 1.60| 127. (84, 28) -4.20| 128. (112, 29) 6.60| 129. (114, 29) -0.50| 130. (24, 30) 6.20| 131. (43, 30) 6.90| 132. (68, 30) 8.90| 133. (76, 30) 5.40| 134. (78, 30) 2.20| 135. (93, 30) -4.90| 136. (99, 30) -1.70| 137. (107, 30) -1.30| 138. (24, 31) 4.60| 139. (105, 31) -5.30| 140. (117, 31) -2.70| 141. (11, 32) -1.70| 142. (18, 32) 7.50| 143. (46, 32) 5.00| 144. (47, 32) 8.60| 145. (66, 32) -1.10| 146. (74, 32) 7.70| 147. (96, 32) -4.70| 148. (101, 32) 2.80| 149. (109, 32) 7.70| 150. (10, 33) -2.70| 151. (55, 33) -0.10| 152. (82, 33) -3.30| 153. (113, 33) 9.40| 154. (26, 34) 1.80| 155. (33, 34) -7.20| 156. (34, 34) 2.10| 157. (35, 34) 0.30| 158. (40, 34) -1.30| 159. (88, 34) 0.20| 160. (16, 35) 7.30| 161. (58, 35) 7.00| 162. (14, 36) -5.80| 163. (45, 36) 1.70| 164. (66, 36) 9.50| 165. (109, 36) 4.10| 166. (9, 37) -6.10| 167. (22, 37) 2.50| 168. (61, 37) 0.10| 169. (120, 37) 6.50| 170. (58, 38) -8.40| 171. (69, 38) -5.50| 172. (94, 38) 7.60| 173. (24, 39) 5.40| 174. (25, 39) -0.30| 175. (43, 39) 6.70| 176. (56, 39) 3.20| 177. (89, 39) -2.10| 178. (93, 39) -6.20| 179. (111, 39) 4.30| 180. (112, 39) -2.70| 181. (7, 40) -0.80| 182. (21, 40) -7.00| 183. (25, 40) -0.80| 184. (30, 40) 0.20| 185. (57, 40) -5.70| 186. (89, 40) 0.40| 187. (15, 41) 4.80| 188. (63, 41) 6.30| 189. (81, 41) -0.10| 190. (83, 41) -8.10| 191. (110, 41) -5.80| 192. (10, 42) 2.20| 193. (81, 42) 1.40| 194. (83, 42) 9.60| 195. (113, 42) -7.10| 196. (5, 43) 9.80| 197. (14, 43) 6.50| 198. (26, 43) 3.60| 199. (39, 43) -8.30| 200. (43, 43) 5.40| 201. (68, 43) -5.40| 202. (92, 43) -8.70| 203. (103, 43) -9.70| 204. (112, 43) -1.40| 205. (60, 44) 9.60| 206. (74, 44) 5.70| 207. (104, 44) 2.20| 208. (10, 45) 7.00| 209. (58, 45) -5.70| 210. (89, 45) -9.70| 211. (91, 45) 0.30| 212. (27, 46) 9.80| 213. (52, 46) -5.70| 214. (90, 46) 6.30| 215. (93, 46) 3.40| 216. (49, 47) -1.10| 217. (52, 47) 9.50| 218. (69, 47) 1.90| 219. (1, 48) -0.30| 220. (32, 48) -8.10| 221. (35, 48) 1.90| 222. (49, 48) 7.20| 223. (52, 48) -9.50| 224. (87, 48) 2.80| 225. (99, 48) -0.30| 226. (100, 48) 6.10| 227. (33, 49) -0.30| 228. (42, 49) 2.70| 229. (47, 49) 5.00| 230. (59, 49) -1.70| 231. (62, 49) -8.70| 232. (102, 49) 0.30| 233. (112, 49) 0.50| 234. (3, 50) -0.80| 235. (9, 50) 7.40| 236. (21, 50) -8.20| 237. (59, 50) 5.20| 238. (85, 50) -4.50| 239. (107, 50) -9.60| 240. (32, 51) 0.50| 241. (47, 51) 6.20| 242. (57, 51) 3.50| 243. (61, 51) 2.00| 244. (120, 51) 9.40| 245. (12, 52) 2.90| 246. (16, 52) -2.80| 247. (27, 52) 5.20| 248. (32, 52) 7.80| 249. (45, 52) -0.40| 250. (59, 52) -2.50| 251. (69, 52) 5.20| 252. (76, 52) -5.10| 253. (91, 52) 9.10| 254. (102, 52) -8.50| 255. (115, 52) 0.30| 256. (1, 53) 8.30| 257. (2, 53) -4.60| 258. (14, 53) -9.70| 259. (29, 53) -2.00| 260. (33, 53) -6.10| 261. (34, 53) 1.20| 262. (49, 53) 3.10| 263. (66, 53) -6.50| 264. (87, 53) -9.70| 265. (94, 53) -1.20| 266. (16, 54) 8.90| 267. (28, 54) 3.70| 268. (34, 54) 9.50| 269. (55, 54) 5.10| 270. (58, 54) 6.00| 271. (90, 54) -7.70| 272. (115, 54) -9.30| 273. (120, 54) 6.40| 274. (22, 55) -7.30| 275. (64, 55) 8.50| 276. (72, 55) 8.30| 277. (79, 55) 8.50| 278. (95, 55) -2.90| 279. (19, 56) -8.90| 280. (22, 56) 4.00| 281. (40, 56) 0.70| 282. (51, 56) -5.10| 283. (60, 56) 4.70| 284. (97, 56) 5.10| 285. (18, 57) -6.70| 286. (72, 57) -8.90| 287. (112, 57) -8.50| 288. (35, 58) 8.70| 289. (80, 58) 0.30| 290. (91, 58) -4.00| 291. (24, 59) 1.80| 292. (27, 59) -5.30| 293. (30, 59) -0.50| 294. (45, 59) 2.50| 295. (47, 59) 8.70| 296. (13, 60) 8.60| 297. (43, 60) -0.70| 298. (89, 60) -6.50| 299. (74, 61) 4.50| 300. (119, 61) -4.50| 301. (83, 62) 5.70| 302. (89, 62) -4.50| 303. (92, 62) -2.00| 304. (116, 62) 9.50| 305. (118, 62) 3.80| 306. (9, 63) -6.00| 307. (14, 63) -0.20| 308. (16, 63) -3.50| 309. (28, 63) -0.60| 310. (47, 63) -3.60| 311. (48, 63) 9.20| 312. (78, 63) -1.40| 313. (105, 63) 5.00| 314. (108, 63) 6.60| 315. (26, 64) 4.60| 316. (65, 64) 2.00| 317. (89, 64) 0.50| 318. (1, 65) -4.00| 319. (3, 65) 7.90| 320. (28, 65) 3.50| 321. (79, 65) 5.50| 322. (80, 65)

1.50| 323. (94, 65) 0.70| 324. (105, 65) 3.60| 325. (73, 66) 3.40| 326. (8, 67) -9.30| 327. (15, 67) -6.50| 328. (23, 67) 9.70| 329. (35, 67) 1.50| 330. (68, 67) 1.90| 331. (78, 67) -7.60| 332. (16, 68) 4.50| 333. (26, 68) -0.20| 334. (27, 68) 3.00| 335. (37, 68) 0.50| 336. (5, 69) 0.60| 337. (8, 69) 5.90| 338. (30, 69) -1.00| 339. (37, 69) -1.80| 340. (40, 69) 1.80| 341. (53, 69) 5.40| 342. (75, 69) -3.00| 343. (88, 69) 6.80| 344. (90, 69) -4.50| 345. (102, 69) -1.30| 346. (103, 69) -4.60| 347. (17, 70) 8.20| 348. (57, 71) -2.20| 349. (74, 71) 1.80| 350. (120, 71) -1.40| 351. (16, 72) -5.50| 352. (76, 72) 0.40| 353. (84, 72) -4.80| 354. (92, 72) 2.40| 355. (105, 72) 2.40| 356. (10, 73) -0.60| 357. (12, 73) -5.90| 358. (13, 73) -6.50| 359. (29, 73) -3.30| 360. (77, 73) 6.10| 361. (99, 73) 6.80| 362. (114, 73) -9.70| 363. (2, 74) -0.30| 364. (103, 74) 7.10| 365. (57, 75) 9.80| 366. (81, 75) 6.00| 367. (101, 75) 6.00| 368. (5, 76) 8.90| 369. (47, 76) 1.60| 370. (71, 76) 3.50| 371. (80, 76) 6.00| 372. (85, 76) -9.30| 373. (94, 76) -2.90| 374. (16, 77) -6.10| 375. (36, 77) -8.10| 376. (42, 77) -3.90| 377. (91, 77) 5.20| 378. (103, 77) -6.30| 379. (29, 78) -6.50| 380. (90, 78) 9.50| 381. (99, 78) 8.50| 382. (14, 79) 3.70| 383. (54, 79) 9.70| 384. (66, 79) 5.30| 385. (89, 79) 6.80| 386. (97, 79) 9.80| 387. (101, 79) -7.60| 388. (116, 79) 8.90| 389. (7, 80) -1.40| 390. (40, 80) -5.80| 391. (90, 80) -7.70| 392. (99, 80) 3.30| 393. (103, 80) -9.00| 394. (119, 80) 8.80| 395. (34, 81) -3.30| 396. (51, 81) 1.20| 397. (60, 81) -9.10| 398. (72, 81) 7.30| 399. (78, 81) 6.80| 400. (85, 81) -1.20| 401. (99, 81) -3.50| 402. (43, 82) -7.40| 403. (55, 82) -4.10| 404. (7, 83) -4.90| 405. (53, 83) -6.50| 406. (78, 83) -0.70| 407. (105, 83) 0.10| 408. (113, 83) 0.80| 409. (1, 84) -5.00| 410. (37, 84) 9.90| 411. (70, 84) -4.60| 412. (90, 84) 3.60| 413. (93, 84) 9.40| 414. (101, 84) 6.20| 415. (7, 85) -5.20| 416. (39, 85) -3.70| 417. (76, 85) -0.80| 418. (79, 85) -5.30| 419. (104, 85) -8.60| 420. (110, 85) 4.80| 421. (18, 86) 4.70| 422. (19, 86) -6.80| 423. (20, 86) -5.10| 424. (28, 86) -7.10| 425. (39, 86) -0.70| 426. (49, 86) 6.00| 427. (67, 86) 9.50| 428. (71, 86) -5.70| 429. (82, 86) 3.90| 430. (114, 86) 4.90| 431. (15, 87) -7.40| 432. (43, 87) 3.20| 433. (33, 88) -6.00| 434. (72, 88) -2.60| 435. (106, 88) -7.90| 436. (17, 89) -3.90| 437. (28, 89) -9.60| 438. (70, 89) 4.20| 439. (108, 89) -6.30| 440. (18, 90) 9.80| 441. (50, 90) 1.10| 442. (58, 90) 3.70| 443. (98, 90) -3.80| 444. (109, 90) -4.40| 445. (28, 91) -7.20| 446. (43, 91) -6.90| 447. (59, 91) -2.60| 448. (70, 91) -4.50| 449. (90, 91) 6.40| 450. (101, 91) 9.40| 451. (117, 91) 8.00| 452. (33, 92) 9.30| 453. (66, 92) 3.30| 454. (91, 92) 8.30| 455. (94, 92) -7.60| 456. (12, 93) -4.80| 457. (41, 93) 2.40| 458. (52, 93) 1.80| 459. (63, 93) 8.00| 460. (102, 93) 8.90| 461. (118, 93) -4.40| 462. (31, 94) -1.90| 463. (4, 95) -5.20| 464. (38, 95) 6.00| 465. (47, 95) -6.00| 466. (63, 95) -2.50| 467. (69, 95) -8.40| 468. (28, 96) -8.50| 469. (31, 96) 3.10| 470. (97, 96) 1.10| 471. (2, 97) 8.40| 472. (15, 97) -4.60| 473. (40, 97) 2.70| 474. (55, 98) 1.40| 475. (86, 98) 9.90| 476. (92, 98) 7.70| 477. (45, 99) -0.30| 478. (66, 99) -6.60| 479. (115, 99) -4.00| 480. (31,100) 5.80| 481. (50,100) 5.00|

411 NONZEROS OF MATRIX W (ROW AND COLUMN INDICES)
ITS DIMENSIONS: 120 180, WITH DENSITY 0.02

1. (10, 1) -6.30| 2. (54, 1) 0.45| 3. (119, 1) -0.40| 4. (50, 2) 2.50| 5. (90, 2) 8.60| 6. (103, 3) 2.90| 7. (30, 4) 5.30| 8. (80, 4) 3.90| 9. (85, 4) 3.50| 10. (98, 4) -4.90| 11. (41, 5) 2.40| 12. (37, 6) 3.85| 13. (75, 6) 1.10| 14. (108, 7) 4.25| 15. (8, 8) 3.70| 16. (72, 8) -6.20| 17. (74, 8) -2.80| 18. (56, 9) 1.70| 19. (70, 9) -4.05| 20. (118, 9) 5.60| 21. (44, 10) 0.60| 22. (105, 10) -8.10| 23. (81, 11) 9.60| 24. (12, 12) 4.60| 25. (85, 12) 4.00| 26. (91, 12) -0.50| 27. (118, 12) -2.80| 28. (13, 13) 5.00| 29. (86, 13) -1.40| 30. (112, 13) 6.90| 31. (28, 14) -6.70| 32. (79, 14)

-0.20| 33. (55, 15) 3.60| 34. (56, 16) 1.20| 35. (90, 16) -4.50| 36. (53, 17) 0.60| 37. (103, 17) 4.70| 38. (41, 18) -3.55| 39. (54, 18) 3.85| 40. (92, 18) 8.70| 41. (48, 19) -1.60| 42. (60, 19) 9.80| 43. (72, 19) 7.60| 44. (52, 20) 10.00| 45. (54, 20) -6.25| 46. (24, 21) 6.10| 47. (1, 22) 8.50| 48. (36, 23) 6.70| 49. (30, 24) 7.50| 50. (6, 25) 2.80| 51. (88, 25) 2.50| 52. (2, 26) -2.50| 53. (4, 26) -4.80| 54. (5, 26) 0.80| 55. (6, 26) -6.05| 56. (8, 26) -7.30| 57. (12, 26) -2.90| 58. (14, 26) -5.70| 59. (17, 26) -7.80| 60. (20, 26) -5.70| 61. (22, 26) -5.30| 62. (23, 26) -7.90| 63. (27, 26) 9.70| 64. (32, 26) -5.90| 65. (34, 26) -4.30| 66. (39, 26) -2.90| 67. (41, 26) 5.35| 68. (47, 26) -5.20| 69. (59, 26) -4.80| 70. (61, 26) -2.90| 71. (62, 26) -5.25| 72. (65, 26) -3.80| 73. (66, 26) -3.90| 74. (71, 26) -1.40| 75. (73, 26) -0.70| 76. (76, 26) -5.00| 77. (77, 26) -1.00| 78. (78, 26) -1.60| 79. (82, 26) -6.45| 80. (83, 26) -1.70| 81. (87, 26) 2.70| 82. (89, 26) 1.70| 83. (92, 26) -1.80| 84. (93, 26) -8.30| 85. (94, 26) -5.50| 86. (95, 26) 0.60| 87. (101, 26) -3.90| 88. (103, 26) -8.60| 89. (104, 26) -8.30| 90. (108, 26) -5.15| 91. (110, 26) -4.00| 92. (111, 26) -4.20| 93. (112, 26) 4.50| 94. (114, 26) 0.30| 95. (115, 26) -5.10| 96. (47, 27) 5.20| 97. (63, 27) 4.00| 98. (11, 28) 2.85| 99. (46, 28) -6.90| 100. (87, 28) 4.10| 101. (33, 29) -7.10| 102. (86, 29) 0.30| 103. (117, 30) -8.30| 104. (46, 31) 5.50| 105. (24, 32) -5.50| 106. (55, 32) -2.50| 107. (11, 33) -6.55| 108. (98, 33) -1.30| 109. (31, 34) 7.80| 110. (79, 34) -6.30| 111. (57, 35) 3.85| 112. (86, 35) 0.10| 113. (120, 35) -0.50| 114. (13, 36) 4.80| 115. (30, 36) -6.90| 116. (119, 37) 3.90| 117. (60, 38) -16.90| 118. (36, 39) 0.50| 119. (50, 39) -9.60| 120. (86, 39) 3.20| 121. (108, 39) 0.90| 122. (116, 39) -9.60| 123. (64, 40) -0.80| 124. (75, 41) 7.80| 125. (21, 42) 4.00| 126. (25, 42) 4.15| 127. (38, 43) -4.80| 128. (16, 44) -3.90| 129. (114, 44) -0.30| 130. (39, 45) 6.10| 131. (56, 45) -5.10| 132. (53, 46) -0.60| 133. (10, 47) 4.60| 134. (27, 47) 0.60| 135. (79, 47) 3.00| 136. (109, 48) -7.40| 137. (113, 48) -0.90| 138. (37, 49) 1.65| 139. (58, 49) 3.20| 140. (110, 49) 4.00| 141. (112, 49) -4.80| 142. (46, 50) 3.80| 143. (74, 50) 1.10| 144. (117, 50) 8.30| 145. (10, 51) 2.40| 146. (74, 51) -1.40| 147. (11, 52) 5.70| 148. (79, 53) 3.50| 149. (90, 53) -4.10| 150. (85, 54) -6.10| 151. (99, 54) -6.90| 152. (26, 55) 3.10| 153. (42, 55) 7.00| 154. (45, 55) -4.35| 155. (87, 55) 1.90| 156. (7, 56) 6.20| 157. (19, 56) -1.00| 158. (17, 57) 7.80| 159. (91, 57) -2.55| 160. (100, 58) 3.00| 161. (48, 59) 1.15| 162. (63, 60) -8.90| 163. (87, 60) -4.90| 164. (57, 61) -9.50| 165. (97, 62) 1.80| 166. (50, 63) 6.00| 167. (9, 64) 1.50| 168. (10, 64) -9.20| 169. (31, 65) -7.80| 170. (67, 65) -1.10| 171. (22, 66) -0.80| 172. (41, 66) -4.20| 173. (62, 66) 4.20| 174. (77, 66) 1.00| 175. (1, 67) -8.60| 176. (15, 67) 0.90| 177. (69, 68) 6.10| 178. (70, 68) -1.80| 179. (111, 69) 4.20| 180. (16, 70) 5.50| 181. (36, 70) -5.00| 182. (11, 71) -0.05| 183. (81, 71) -2.00| 184. (89, 71) 7.50| 185. (32, 72) 5.90| 186. (86, 72) -2.50| 187. (97, 73) -8.30| 188. (44, 74) 0.50| 189. (83, 75) 7.80| 190. (119, 75) 0.60| 191. (16, 76) 6.80| 192. (3, 77) 1.50| 193. (44, 77) -9.70| 194. (56, 77) -0.10| 195. (63, 77) 0.90| 196. (74, 77) 0.50| 197. (75, 77) -0.30| 198. (33, 78) 4.40| 199. (100, 78) -0.20| 200. (115, 78) 5.10| 201. (98, 79) 6.20| 202. (66, 80) -0.40| 203. (72, 80) 1.40| 204. (12, 81) -1.70| 205. (30, 81) -6.70| 206. (57, 81) 2.15| 207. (60, 81) 7.10| 208. (100, 81) -9.20| 209. (103, 81) 1.00| 210. (48, 82) 2.65| 211. (93, 82) 8.30| 212. (51, 83) 1.50| 213. (67, 84) 1.10| 214. (81, 84) -3.60| 215. (7, 85) -8.70| 216. (30, 85) 3.70| 217. (65, 85) 3.80| 218. (109, 85) 7.60| 219. (70, 86) 3.50| 220. (62, 87) 1.05| 221. (45, 88) -6.15| 222. (75, 88) -3.30| 223. (57, 89) 1.55| 224. (28, 90) 7.90| 225. (33, 90) -2.20| 226. (81, 91) 1.20| 227. (49, 92) 2.70| 228. (6, 93) 3.05| 229. (24, 93) -0.60| 230. (43, 93) 7.80| 231. (44, 93) 8.60| 232. (70, 93) -3.35| 233. (92, 93) -6.90| 234. (96, 93) 3.90| 235. (120, 93) -0.50| 236. (34, 94) 3.70| 237. (50, 94) -4.60| 238. (101, 94) 3.90| 239. (88, 95) 4.10| 240.

(52, 96) 5.40| 241. (107, 96) 7.50| 242. (107, 97) 0.40| 243. (104, 98) 8.30| 244. (107, 98) -8.40| 245. (58, 99) 3.60| 246. (22,100) 2.00| 247. (26,100) 3.55| 248. (78,100) 1.60| 249. (63,101) 4.90| 250. (119,101) -0.70| 251. (25,102) 1.15| 252. (63,102) -4.00| 253. (87,102) -3.80| 254. (106,102) 3.00| 255. (21,103) -5.90| 256. (51,103) 8.00| 257. (7,104) 4.50| 258. (9,104) -2.70| 259. (51,104) 0.80| 260. (5,105) 1.40| 261. (28,105) -1.20| 262. (39,105) -3.20| 263. (69,105) -6.10| 264. (109,106) -0.20| 265. (33,107) 4.90| 266. (105,108) 1.00| 267. (30,109) -2.90| 268. (51,109) -5.50| 269. (4,110) 4.80| 270. (66,110) 4.30| 271. (68,110) 0.80| 272. (52,111) 4.00| 273. (55,111) 2.50| 274. (16,112) -8.40| 275. (54,112) 1.95| 276. (113,112) 0.90| 277. (15,113) -0.90| 278. (119,113) -3.40| 279. (37,114) -3.00| 280. (51,115) -4.80| 281. (116,115) 5.40| 282. (18,116) 0.30| 283. (81,117) 8.50| 284. (105,117) 7.10| 285. (2,118) 2.50| 286. (7,118) -2.00| 287. (49,118) -1.90| 288. (29,119) 9.80| 289. (59,120) 4.80| 290. (99,120) 3.00| 291. (1,121) 0.10| 292. (120,121) 1.00| 293. (8,122) 3.60| 294. (42,122) 2.70| 295. (64,122) 9.40| 296. (71,122) 1.40| 297. (100,123) 6.40| 298. (64,124) 0.60| 299. (38,125) 1.00| 300. (45,125) 4.25| 301. (20,126) 2.40| 302. (85,126) -1.40| 303. (84,127) -4.90| 304. (40,128) 8.20| 305. (42,129) -4.70| 306. (118,129) 5.70| 307. (3,130) -1.50| 308. (42,131) -6.80| 309. (68,131) 0.55| 310. (76,131) 1.90| 311. (80,131) -3.00| 312. (82,131) 2.40| 313. (89,131) -9.20| 314. (27,132) -10.30| 315. (29,132) 0.30| 316. (50,132) 5.70| 317. (96,133) 2.10| 318. (106,133) -3.00| 319. (48,134) -6.00| 320. (36,135) 4.50| 321. (61,135) 2.90| 322. (82,135) 4.05| 323. (5,136) -2.20| 324. (14,136) 5.70| 325. (49,136) 0.70| 326. (52,136) -9.70| 327. (94,136) 3.90| 328. (46,137) -2.40| 329. (91,138) 0.20| 330. (56,139) 2.30| 331. (52,140) -9.70| 332. (68,140) 1.15| 333. (35,141) -3.30| 334. (58,141) -6.80| 335. (80,142) 4.15| 336. (18,143) 3.80| 337. (11,144) 3.75| 338. (84,145) 4.90| 339. (94,145) -2.15| 340. (43,146) -7.80| 341. (29,147) -0.30| 342. (57,148) 1.95| 343. (72,148) 4.10| 344. (75,148) -8.50| 345. (20,149) 3.30| 346. (38,149) 3.80| 347. (6,150) 0.20| 348. (18,150) 4.00| 349. (19,150) 1.00| 350. (81,150) -0.80| 351. (97,150) 6.50| 352. (35,151) 3.30| 353. (81,152) -9.10| 354. (99,152) -0.25| 355. (116,152) 4.10| 356. (107,153) 0.50| 357. (45,154) 3.35| 358. (96,154) -6.60| 359. (118,154) -8.50| 360. (13,155) -9.80| 361. (80,155) -5.05| 362. (75,156) 3.20| 363. (76,156) 3.10| 364. (116,156) 0.10| 365. (74,157) 0.30| 366. (99,158) 4.15| 367. (91,159) -3.20| 368. (102,159) 5.70| 369. (26,160) -6.65| 370. (86,161) 0.30| 371. (10,162) 8.50| 372. (18,163) -7.80| 373. (49,163) 2.10| 374. (44,164) 5.30| 375. (22,165) 4.10| 376. (25,166) -5.30| 377. (40,166) -8.20| 378. (88,167) -6.25| 379. (23,168) 7.90| 380. (48,168) -0.50| 381. (72,168) -6.90| 382. (73,169) 0.70| 383. (88,169) -0.35| 384. (70,170) 5.70| 385. (91,170) 6.05| 386. (57,171) 6.70| 387. (80,171) -6.50| 388. (85,171) -7.40| 389. (21,172) 0.10| 390. (48,172) 4.30| 391. (9,173) 1.20| 392. (69,174) 9.00| 393. (102,174) -5.70| 394. (45,175) 2.90| 395. (63,175) 3.10| 396. (64,175) -9.20| 397. (69,175) -9.00| 398. (83,175) -6.10| 399. (95,175) -0.60| 400. (68,176) 4.00| 401. (81,176) 5.80| 402. (94,176) 3.75| 403. (37,177) -2.50| 404. (68,177) -6.50| 405. (42,178) 1.80| 406. (21,179) 1.80| 407. (34,179) 0.60| 408. (49,179) -3.60| 409. (74,179) 2.30| 410. (112,179) -6.60| 411. (96,180) 0.60|

OBJECTIVE FUNCTION OF THE SECOND STAGE PROBLEM

3.230 9.180 6.600 1.240 7.550 8.630 3.280 8.050 4.770 3.380 5.950 1.270 8.660 5.590 1.620 7.590 9.500 4.610 2.990 7.650 1.180 9.200 2.800 4.520 9.370 6.420 0.080 2.330 1.080 8.030 9.410 0.850 0.980 1.430 9.200 1.240 5.480 7.140 8.960 7.240 8.330 1.800 1.520 3.450 1.930 0.700 3.460 7.770 1.990 4.280 4.480 3.670 6.110 9.620 0.220 4.240 0.440 7.700 5.930 1.470 7.420 1.690 6.820 1.770

1.480 8.230 1.670 1.260 8.370 3.890 8.370 7.270 6.490 5.350 3.090 8.120 1.220 6.380 4.820 6.240
3.990 8.190 4.470 9.590 4.830 8.920 8.280 6.110 5.580 7.920 8.010 9.240 3.370 5.170 6.860 5.720
2.240 8.120 1.840 7.310 2.580 2.670 2.630 7.860 7.210 7.180 8.790 3.460 5.330 1.790 2.480 9.680
3.290 9.630 1.640 3.240 5.250 7.170 7.650 3.580 6.990 1.280 8.560 5.930 9.470 8.490 3.720 4.320
5.340 3.310 0.780 7.070 7.040 4.690 5.810 6.060 2.950 7.860 5.910 1.600 1.080 1.600 5.310 4.450
5.870 9.620 6.890 7.540 1.220 1.380 7.010 8.590 0.630 1.440 3.100 5.280 1.600 6.210 3.840 0.370
4.500 4.300 9.740 0.600 0.110 0.860 8.930 9.760 7.870 9.070 2.590 8.040 2.120 8.580 8.020 1.670
3.440 0.100 8.210 5.700

EXPECTED VALUE OF RANDOM RHS OF THE SECOND STAGE PROBLEM

-0.40 4.40 4.10 -2.30 1.00 -1.30 -1.30 4.30 -0.10 -1.10 -2.60 4.10 0.80 2.70 4.60 -2.10 1.30 1.00 0.80
-1.60 4.10 0.50 -2.50 -1.30 3.70 -2.10 3.90 2.70 -4.20 -3.50 2.90 2.90 0.80 4.40 -3.60 -4.80 0.20 1.90
2.40 -2.70 0.80 3.90 -4.60 -0.70 -2.30 -0.10 2.90 3.60 3.10 0.50 1.40 -1.20 2.70 4.00 1.50 4.00 -4.60
-1.40 -2.60 -3.30 -0.90 -4.80 3.90 3.80 -2.90 2.70 -3.70 1.50 2.70 1.20 -2.40 -3.50 -4.60 4.40 -1.80
1.60 -3.10 3.50 4.20 -3.10 4.80 -3.20 3.20 0.50 1.70 2.90 -4.60 -3.10 -3.50 1.20 -2.70 2.70 1.40 -3.70
0.90 3.60 0.50 4.20 3.50 -2.90 -3.80 4.20 -4.70 1.30 -2.80 1.90 -0.20 -3.10 4.10 -0.40 2.70 2.80 2.70
-4.40 -3.70 -1.70 -4.00 4.80 3.90 -1.70

STANDARD DEVIATION OF RIGHT HAND SIDE VECTOR

0.08 0.88 0.82 0.46 0.20 0.26 0.26 0.86 0.02 0.22 0.52 0.82 0.16 0.54 0.92 0.42 0.26 0.20 0.16 0.32
0.82 0.10 0.50 0.26 0.74 0.42 0.78 0.54 0.84 0.70 0.58 0.58 0.16 0.88 0.72 0.96 0.04 0.38 0.48 0.54
0.16 0.78 0.92 0.14 0.46 0.02 0.58 0.72 0.62 0.10 0.28 0.24 0.54 0.80 0.30 0.80 0.92 0.28 0.52 0.66
0.18 0.96 0.78 0.76 0.58 0.54 0.74 0.30 0.54 0.24 0.48 0.70 0.92 0.88 0.36 0.32 0.62 0.70 0.84 0.62
0.96 0.64 0.64 0.10 0.34 0.58 0.92 0.62 0.70 0.24 0.54 0.54 0.28 0.74 0.18 0.72 0.10 0.84 0.70 0.58
0.76 0.84 0.94 0.26 0.56 0.38 0.04 0.62 0.82 0.08 0.54 0.56 0.54 0.88 0.74 0.34 0.80 0.96 0.78 0.34

RESULTS OF A COMPUTER RUN

MAK-MORTON-WOOD: LOWER BOUND (STD.DEV) 0.3691D+04 0.56D+02
NBATCH,NBLOCS,MM,MN,NOZEROS 100 24 2931 4420 26162
TTT: MMW LOWER BOUND TIME 198.8 NBLOCS= 24 NBATCH= 100
AVERAGE OF FEASIBLE SOLUTIONS IS:

0.000 0.000 0.195 1.433 0.130 0.329 0.039 0.083 0.000 0.000 0.000 0.142 0.000 0.000 0.628 0.000
0.000 1.136 0.594 1.161 0.000 0.000 3.141 0.000 0.173 0.863 0.101 0.000 0.016 0.176 0.000 0.000
0.587 0.252 0.000 0.000 0.114 0.068 0.122 0.000 0.941 0.102 0.000 0.000 0.029 0.000 0.020 0.017
0.000 0.056 0.195 0.000 0.040 0.234 0.010 0.257 0.000 0.000 0.162 0.189 0.024 0.000 0.682 0.000
0.000 0.135 0.086 0.000 0.207 0.314 0.121 0.000 0.000 0.000 0.000 0.004 0.004 0.297 0.000 0.000
0.129 0.017 0.000 0.000 0.154 0.159 0.020 0.000 0.316 0.045 0.161 0.309 0.105 0.079 0.505 0.000
0.000 0.050 0.000 0.000

TTT: PRODUCING FEAS. SOL.S AND BOUNDS 749.2 NO. OF FEAS. POINTS 1204
CONSTANTS: KINIT = 6554 FIRST - SECOND PHASE 78648 19662
MAXITER = 19800 IREINV,KMETHOD,KSIZE = 5000 2 5
TTT: Q-FUNCTION EVALUATION FOR 6554 POINTS 254.66 AVERAGE TIME= 0.039
MAX AND MIN FEAS.POINTS(SD) 0.673D+04 3 7.85 0.348D+04 913 24.69
TTT: UPDATING BY XANINIT TAKES TIME: 5709.770

```
TTT: INVERSION (COLQINV) TIME IS = 4744.819
KMETHOD AND KSIZE OF MC INTEGRATION 2 15
TTT: UPDATING (LAM,QM) TIME: 3.170 KACT 6560
TTT: INVERSION (COLQINV) TIME IS = 4781.629
MAININD(KACT),MAINKACT,KMETHOD,KSIZE 12005 912 2 15

   K XPOINT | QVALUE (SDEV),LIN+QFUNC, | OLD AND LAST QUAD. APPR.
   6555 0.000 0.000 0.000 | 9499.31 ( 3.90)9499.3246 | 0.950D+04 0.946D+04
   ....
   7557 0.000 0.000 0.000 | 6155.25 (22.09)6155.3111 | 0.616D+04 0.630D+04
   ....
   8559 0.000 0.000 0.000 | 6483.95 (62.56)6484.0285 | 0.647D+04 0.647D+04
   ....
   9561 0.000 0.000 0.000 | 6874.83 (14.83)6874.7921 | 0.685D+04 0.687D+04
   ....
   10557 0.000 0.000 0.000 | 8061.96 ( 3.86)8061.9388 | 0.696D+04 0.811D+04
   ....
   11565 0.000 0.000 0.000 | 5267.60 (95.10)5267.6426 | 0.525D+04 0.526D+04
   ....
   11859 0.000 0.000 0.096 | 4715.50 (78.08)4715.4820 | 0.463D+04 0.463D+04
   ....
   11955 0.000 0.000 0.212 | 3811.99 (31.35)3811.9585 | 0.383D+04 0.383D+04
   11960 0.000 0.000 0.183 | 3694.12 (45.68)3694.1139 | 0.369D+04 0.370D+04
   11965 0.000 0.000 0.191 | 3704.25 (43.90)3704.2226 | 0.368D+04 0.369D+04
   11970 0.000 0.000 0.174 | 3775.11 (39.31)3775.0901 | 0.371D+04 0.370D+04
   11975 0.000 0.000 0.178 | 3772.12 (32.32)3772.0907 | 0.374D+04 0.374D+04
   11980 0.000 0.000 0.189 | 3754.93 (77.09)3754.8953 | 0.368D+04 0.368D+04
   11985 0.000 0.000 0.188 | 3746.47 (50.13)3746.4397 | 0.368D+04 0.368D+04
   11990 0.000 0.000 0.189 | 3720.93 (52.61)3720.9059 | 0.368D+04 0.368D+04
   11995 0.000 0.000 0.190 | 3759.64 (68.64)3759.6082 | 0.368D+04 0.368D+04
   12000 0.000 0.000 0.191 | 3723.04 (61.27)3723.0118 | 0.368D+04 0.368D+04
   12005 0.000 0.000 0.192 | 3714.38 (44.61)3714.3573 | 0.368D+04 0.368D+04
   AT END, DIFFICULTY= 0.140 O.G. VALUE AND ST.DEV 0.373D+04 0.175D+03

   * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
   * SRA TIME IN SEC-S 30351 WHERE KACT= 12005, FUNCVAL= 0.371D+04 *
   * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *

THE DIFFICULTY: 0.149 FOR A 100 120 PROBLEM
MMW LOWER BOUND (SD) 0.3691D+04 (+- 5.63), TEMPMIN (SD)= 0.3652D+04 (+- 1.37)
TTT: TIME OF EVALUATING ACCURATE FUNC.VAL 214.2
   LAST XPOINT 12005 FUNCVAL =3709.58136 (+- 0.279) QUAD APPR= 0.368D+04
```

0.000000 0.000000 0.191534 1.453430 0.132095 0.328934 0.037922 0.082929 0.000000 0.000000
0.000000 0.140758 0.000000 0.000000 0.574566 0.000000 0.000000 1.178090 0.492692 1.204865
0.000000 0.000000 3.160034 0.000000 0.172704 0.862969 0.101884 0.000000 0.015866 0.153803
0.000000 0.000000 0.507538 0.241025 0.000000 0.000000 0.113913 0.068357 0.120547 0.000000
0.939948 0.100683 0.000000 0.000000 0.028485 0.000000 0.019674 0.018137 0.000000 0.056034
0.193600 0.000000 0.038039 0.233874 0.010073 0.256697 0.000000 0.000000 0.162378 0.189223
0.024318 0.000000 0.674747 0.000000 0.000000 0.135058 0.086190 0.000000 0.184552 0.314194
0.118042 0.000000 0.000000 0.000000 0.000000 0.006854 0.003747 0.298651 0.000000 0.000000
0.128723 0.015660 0.000000 0.000000 0.155179 0.158679 0.022023 0.000000 0.315640 0.044810
0.160562 0.304202 0.107833 0.076569 0.514862 0.000000 0.000000 0.050109 0.000000 0.000000
T O T A L RUNNING TIME IN SEC-S 31315.3
MMW CONFIDENCE: POINT EST. (SD) 0.208D+02 (+- 1.11), PERCENT: 0.56

Irodalomjegyzék

[AM 79] Anderson, B.D.O., Moore, J.B.: Optimal filtering, Prentice-Hall, Englewood Cliffs, 1979.

[AF 01] Anderson, E.J., Ferris, M.C.: A direct search algorithm for optimization with noisy function evaluations, SIAM J. Optim. 11 (2001) 837-857.

[BD 93] Bálint, E., Deák, I.: Párhuzamos számítógépek: optimalizálási programok, Alk. Mat. Lapok 17 (1993) 1-18.

[BSS 94] Bazaraa, M.S., Sherali, H.D., Shetty, C.M.: Nonlinear Programming – Theory and Algorithms, Wiley, 1994.

[BMP 90] Benveniste, A., Metivier, M., Priouret, P.: Adaptive algorithms and stochastic approximation, Springer, in: Applications of Mathematics, Vol. 22., 1990, pp. 365.

[BT 89] Bertsekas, D.P., Tsitsiklis, J.N.: Parallel and distributed computation – Numerical methods, Prentice Hall, 1989.

[BD 91] Birge, J., Dulà, J.H.: Bounding separable recourse functions with limited distribution information, Annals of Operations Research 30 (1991) 277-298.

[BL 97] Birge, J., Louveaux, F.: Introduction to stochastic programming, Springer, 1997.

[Bje 90] Bjerager, P.: On computation methods for structural reliability analysis, Structural Safety 9 (1990) 79-96.

[Bjo 96] Bjorck, A.: Numerical methods for least squares problems, SIAM, 1996.

[BD 87] Box, G.E.P., Draper, N.R.: Empirical model-building and response surface, J. Wiley and Sons, 1987.

[Bre 94] Breslaw, J.A.: Evaluation of multivariate normal probability integrals using low variance simulator, Review of Econom. Stat. 76 (1994) 673-682.

[BP 01] Bukszár, J., Prékopa, A.: Probability bounds with cherry trees, Mathematics of Operations Research 26 (2001) 174-192.

[BS 02] Bukszár, J., Szántai, T.: Probability bounds given by hyper-cherry trees, Optimization Methods and Software 17 (2002) 409-422.

[CH 86] Collings, B.J., Hembree, G.B.: Initializing generalized feedback shift register pseudorandom number generators, J. ACM 33 (1986) 706-711.

[Cor 81] Cornell, J.A.: Experiments with mixtures: designs, models, and the analysis of mixture data, J. Wiley and Sons, 1981.

[Dan 55] Dantzig, G.B.: Linear programming under uncertainty, Management Science 1 (1955) 197-206.

[De 72] Deák, I.: Egy sztochasztikus programozási modell számítógépes kiértékelése, MTA Számítástechnikai Központ Közleményei 9 (1972) 33-49.

[De 76] Deák, I.: A többdimenziós tér halmazai valószínűségeinek kiszámítása normális eloszlás esetén, Alkalmazott Mat. Lapok 2 (1976) 17-26.

[De 78] Deák, I.: Monte Carlo módszerek a többdimenziós térben elhelyezkedő halmazok valószínűségének meghatározására normális eloszlás esetén, Alkalmazott Mat. Lapok 4 (1978) 35-94.

[De 79] Deák, I.: Computation of multiple normal probabilities, Recent results in Stochastic Programming (eds. P.Kall, A.Prékopa), Springer Lecture Notes in Economics and Math. Systems V. 179., Springer Verlag, Berlin-New York, 107-120.

[De 80a] Deák, I.: Three digit accurate multiple normal probabilities, Numerische Mathematik 35 (1980) 369-380.

[De 80b] Deák, I.: Monte Carlo módszerek a többdimenziós térben elhelyezkedő halmazok valószínűségének meghatározására normális eloszlás esetén, kandidátusi értekezés, MTA 1980.

[De 81] Deák, I.: An economical method for random number generation and a normal generator, Computing 27 (1981) 113-121.

[De 86a] Deák, I.: The economical method for generating random samples from discrete distributions, ACM TOMS 12 (1986) 34-36.

[De 86b] Deák, I.: Computing probabilities of rectangles in case of multinormal distributions, J. Statist. Comput. and Simul. 26 (1986) 101-114.

[De 86c] Deák, I.: Véletlenszámgenerátorok és alkalmazásuk, in: Az operációkutatás matematikai módszerei (series editor A. Prékopa), Akadémiai Kiadó, Budapest, 1986, 235 o.

[De 88] Deák, I.: Multidimensional integration and stochastic programming, in: Numerical techniques for stochastic optimization 1984, Springer series in computational mathematics, (eds. Yu.Ermoliev, R.Wets.), Springer Verlag, 1988, 187-200.

[De 89] Deák, I.: Procedures to solve STABIL on a parallel computer, Working Paper, University of Wisconsin, Department of Industrial Engineering, 89-10, 1989.

[De 90a] Deák, I.: Random number generators and simulation, in: Mathematical Methods of Operations Research (series editor A.Prékopa), Akadémiai Kiadó (Publishing House of the Hungarian Academy of Sciences), Budapest, 1990.

[De 90b] Deák, I.: Uniform random number generators for parallel computers, Parallel Computing 15 (1990) 155-164.

[De 96] Deák, I.: Asynchronous parallel models for optimization, Proceedings of the ParNum96, International Workshop on Parallel Numerics, Slovenia, Gozd Martuljek, 1996 (eds. R.Trobec, M.Vajtersic, P.Zinterhof, J.Silc, B.Robic), J.Stefan Institute – Ljubljana, 195-205.

[De 97a] Deák, I.: Approximations to the standard normal distribution function, Central European Journal on Operations Research and Economics 5 (1997) 283-293.

[De 97b] Deák, I.: Variants of a general parallel optimization framework, Proc. of the International Workshop Parallel Numerics '97, Zakopane, Poland, (eds. R.Wyrzykowski, H.Piech, B.Mochnacki, M.Wajtersic, P.Zinterhof) 1997, 143-154.

[De 98a] Deák, I.: Regression estimators for multinormal distributions: computer experiences in root finding, in: Stochastic programming methods and technical applications, (eds. K. Marti, P. Kall), Lecture Notes in Economics and Mathematical Systems, Springer, V. 458, 1998, 279-293.

[De 98b] Deák, I.: Linear regression estimators for multinormal distributions in optimization of stochastic programming models, European Journal of Operational Research 111 (1998) 555-568.

[De 98c] Deák, I.: Normal probabilities – computer experiences and program description, University of Zurich, Institut of Operations Research, Manuskripte, 1998.

[De 98d] Deák, I.: Evaluating roots: Computer experiences and program description, University of Zurich, Institut of Operations Research, Manuskripte, 1998.

[De 99] Deák, I.: Iteration grain sized asynchronous parallel algorithms in optimization, International J. of Computer Mathematics, 71 (4) 1999, 183-196.

[De 00] Deák, I.: Subroutines for computing normal probabilities of sets – computer experiences, Annals of Operations Research V. 100 (2000) 103-122.

[De 01a] Deák, I.: Successive regression approximations for solving equations, Pure Mathematics and Applications 12 (2001) 25-50.

[De 01b] Deák, I.: Computer experiences with successive regression approximations for solving equations, Optimization Theory (eds. F. Gianessi, P. Pardalos, T. Rapcsák) Kluwer Academic Publishers, Dordrecht-Boston-London, 2001, 65-80.

[De 02] Deák, I.: Computing two-stage stochastic programming problems by successive regression approximations, Stochastic Optimization Techniques – Numerical Methods and Technical Applications (ed. K. Marti) Springer LNEMS V. 513 (2002) 91-102.

[De 03a] Deák, I.: Probabilities of simple n-dimensional sets for the normal distribution, IIE Transactions (Operations Engineering) 35 (2003) 285-293.

[De 03b] Deák, I.: Two-stage stochastic problems with correlated normal variables: computational experiences, Annals of Operations Research, 2002, megjelenés alatt.

[De 03c] Deák, I.: Solving stochastic programming problems by successive regression approximations – Numerical results, in: Dynamic Stochastic Optimization (eds. K.Marti, Y. Ermoliev, G. Pflug) Springer LNEMS V. 532 (2003) 209-224.

[De 03d] Deák, I.: Bevezetés a sztochasztikus programozásba, BKÁE Operációkutatás No. 3., Aula kiadó, 2003, 118 o.

[De 04] Deák, I.: Testing successive regression approximation by large two-stage problems, előkészületben az Annals of Operations Research, Tucson, Arizona kötetbe.

[DP 88] DeMatteis, A, Pagnutti, S.: Parallelization of random number generators and long-range correlation, Numerische Mathematik 53 (1988) 595-608.

[Dem 68] Dempster, M.A.H.: On stochastic programming I.: Static linear programming under risk, Mathematical Analysis and its Applications 21 (1968) 304-343.

[Dem 88] Dempster, M.A.H.: On stochastic programming II.: Dynamic problems under risk, Stochastics 25 (1988) 15-42.

[DB 89] Ditlevsen, O. J., Bjerager, P.: Plastic reliability analysis by directional simulation, J. Engineering Mechanics V. 115. (1989) 1347-1362.

[DS 66] Draper, N.R., Smith, H.: Applied regression analysis, John Wiley and Sons, 1966.

[DC 86] Ducrocq, V., Colleau, J.J.: Interest in quantitative genetics of Dutt's and Deak's methods for numerical computation of multivariate normal probability integrals, Genetique, Selection et Evolution 18 (1986) 447-474.

[Dvo 56] Dvoretzky, A.: On stochastic approximation, 3rd Berkeley symposium on Math. Stat. and Probability, Vol. I. (ed. J. Neyman) Univ. of California Press, 1956.

[Erm 76] Ermoliev, Yu.M.: Methods of stochastic programming, in: Optimization and Operations Research, Nauka, 1976 (in Russian).

[Erm 83] Ermoliev, Yu.M.: Stochastic quasigradient methods and their applications to systems optimization, Stochastics 9 (1983) 1-36.

[Fab 00] Fábián, C.I.: Bundle type methods for inexact data, Central European Journal on Operations Research and Economics 8 (ed. T.Csendes, T. Rapcsák, special issue) 35-55.

[FS 03] Fábián, C.I.,Szőke, Z.: Solving two-stage stochastic programming problems with the level decomposition method, Rutcor Research Report, RRR 35-2003.

[Fox 97] Fox, J.: Applied regression analysis, models, and related methods, Sage publications 1997.

[Fra 92] Frauendorfer, K.: Stochastic two stage programming, Lecture Notes in Economics and Mathematical Systems, V. 392, Springer, 1992.

[Fre 86] Frenkel, K.A.: Evaluating two massively parallel machines, Comm. ACM 29 (1986) 752-758.

[Fus 88] Fushimi, M.: Designing a uniform random number generator whose subsequences are k-distributed, SIAM J. Computing 17 (1988) 89-99.

[Gai 86] Gaivoronski, A.: Stochastic quasigradient methods and their implementation, in: Numerical techniques for stochastic optimization 1984, Springer series in computational mathematics, (eds. Yu. Ermoliev, R. Wets.), Springer Verlag, 1988, 313-351.

[GZ 86] Gassmann, H., Ziemba, W.T.: A tight upper bound for the expectation of a convex function of a multivariate random variable, Math. Programming Study 27 (1986) 39-53.

[Ga 88] Gassmann, H.: Conditional probability and conditional expectation of a random vector, in: Numerical techniques for stochastic optimization (eds. Yu. Ermoliev, R. Wets), Springer series in computational mathematics, Springer Verlag, 1988, 237-254.

[GDS 02] Gassmann, H., Deák, I., Szántai, T.: Computing multivariate normal probabilities: a new look, J. Computational and Graphical Statistics 11 (2002) 920-949.

[Ger 92] Gerencsér, L.: Rate of convergence of recursive estimators, SIAM J. Control and Optimization 30 (1992) 1200-1227.

[GW 96] Györfi, L., Walk, H.: On the averaged stochastic approximations for linear regression, SIAM J. Control and Optimization 34 (1996) 31-61.

[HFR 96] Hajivassiliou, V., McFadden, D., Ruud, P.: Simulation of multivariate normal rectangle probabilities and their derivatives, Journal of Econometrics 72 (1996) 85-134.

[HZ 74] Hammer, P.L., Zoutendijk, G.(eds.): Mathematical programming: theory and practice, North Holland – American Elsevier, 1974.

[HH 64] Hammersley, J.M., Handscomb, D.C.: Monte Carlo methods, Methuen, London, 1964.

[HS 96] Higle, J.L., Sen, S.: Stochastic decomposition: a statistical method for large scale stochastic linear programming, Kluwer Academic Publishers, 1996.

[Inf 94] Infanger, G.: Planning under uncertainty: solving large scale stochastic linear programs, Boyd and Fraser Publ. Co., Danvers, MA, 1994.

[JK 72] Johnson, N.L., Kotz, S.: Distributions in Statistics, vol.I.–IV., J. Wiley, 1972.

[Kal 76] Kall, P.: Stochastic Linear Programming, Springer, 1976.

[KW 94] Kall, P., Wallace, S.: Stochastic programming, Wiley, 1994.

[KM 96] Kall, P., Mayer, J.: SLP-IOR: An interactive model management system for stochastic linear programs, Math. Programming 75 (1996) 221-240.

[KRF 88] Kall, P., Ruszczyński, A., Frauendorfer, K.: Approximation techniques in stochastic programming, in: Numerical techniques for stochastic optimization (eds. Yu. Ermoliev, R. Wets), Springer series in computational mathematics, 1988, 33-64.

[KLS 97] Kannan, R., Lovász, L., Simonovits, M.: Random walks and an $O^*(n^5)$ volume algorithm, Random Struct. Algorithms 11 (1997) 1-50.

[KC 87] Khuri, A.I., Cornell, J.A.: Response surfaces: design and analyses, Marcell Dekker, New York-Basel-Milwaukee, 1987.

[KK 96] Kibzun, A.I., Kan, Y.S.: Stochastic programming problems (with probability and quantile functions), J. Wiley - Interscience Series in Systems and Optimization, 1996.

[KW 52] Kiefer, J., Wolfowitz, J.: Stochastic estimation of a regression function, Ann. Math. Stat. 23 (1952) 462-466.

[KBJ 00] Kotz, S., Balakrishnan, N., Johnson, N.L.: Continuous multivariate distributions, II. edition, Wiley series in probability and statistics, 2000.

[Kha 95] Klein Haneveld, W.K.: Duality in stochastic linear and dynamic programming, Lecture Notes in Economics and Mathematical Systems, Springer, V. 274., 1995.

[Kom 87] Komáromi, É.: On properties of the probabilistic constrained linear programming problem and its dual, J. Optimization Theory and Applications 55 (1987) 337-390.

[KP 79] Kronmal, R.A., Peterson, A.V.: On the alias method for generating random variables from a discrete distribution, Amer. Stat. 33 (1979) 214-218.

[KC 78] Kushner, H.J., Clark, D.S.: Stochastic approximation method for constrained and unconstrained systems, Springer, in: Applied Mathematical Sciences 26, 1978.

[KY 93] Kushner, H.J., Yang, J.: Stochastic approximation with averaging of the iterates: optimal asymptotic rate of convergence for general processes, 1993, manuscript.

[KY 95] Kushner, H.J., Yang, J.: Stochastic approximation with averaging and feedback: rapidly convergent „on-line" algorithms, IEEE Trans. on Automatic Control 40 (1995) 24-34.

[LR 81] Lai, T.L., Robbins, H.: Consistency and asymptotic efficiency of slope estimates in stochastic approximation schemes, Z. Wahrscheinlichtkeittheorie und verw. Gebiete 56 (1981) 329-360.

[LH 95] Lawson, C.L., Hanson, R.J.: Solving least squares problems, SIAM Classics in Applied Mathematics, 1995.

[LP 73] Lewis, T.G., Payne, W.H.: Generalized feedback shift register pseudorandom number algorithm, J. ACM 20 (1973) 456-468.

[LN 83] Lidl, R., Niederreiter, H.: Finite fields, Addison-Wesley, Reading, MA, 1983.

[Loh 93] Lohr, S. L.: Multivariate normal probabilities of star shaped regions, Appl. Stat. 42 (1993) 576-582.

[Lov 99] Lovász, L.: Hit-and-run mixes fast, Math. Programming, Ser. A. 86 (1999) 443-461.

[LS 93] Lovász, L., Simonovits, M.: Random walks in a convex body and an improved volume algorithm, Random Struct. Algorithms 4 (1993) 359-412.

[LR 88] Lootsma, F.A., Ragsdell, K.M.: State of the art in parallel nonlinear optimization, Parallel Computing 1988 (6) 131-155.

[Lue 69] Luenberger, D.G.: Optimization by vector space methods, Reading MA., Addison-Wesley, 1969.

[Lue 84] Luenberger, D.G.: Linear and nonlinear programming, Reading MA., Addison-Wesley, 1984.

[MMW 99] Mak, W.K., Morton, D.P., Wood, R.K.: Monte Carlo bounding techniques for determining solution quality in stochastic programs, Operations Research Letters 24 (1999) 47-56.

[MT 85] Marsaglia, G., Tsay, L.-H.: Matrices and the structure of random number sequences, Linear Algebra Appl. 67 (1985) 147-156.

[Mar 88] Marti, K.: Descent direction and efficient solutions in discretely distributed stochastic programs, Lecture Notes in Economics and Mathematical Systems, Springer, 1988.

[Mar 92] Marti, K.: Semi-stochastic approximation by the response surface methodology (RSM), Optimization 25 (1992) 209-230.

[May 92] Mayer, J.: Computational techniques for probabilistic constrained optimization problems, in: Lecture Notes on Economics and Mathematical Systems, Springer, V. 379. (1992) 141-164.

[May 98] Mayer, J.: Stochastic linear programming algorithms, Gordon and Breach, 1998.

[MG 97] Monahan J., Genz, A.: Spherical-radial integration rules for Bayesian computation, J. Amer. Stat. Ass. V. 92. (1997) 664-674.

[Nie 87] Niederreiter, H.: A statistical analysis of generalized feedback shift register pseudorandom number generators, SIAM J. Sci. Statistical Computing 8 (1987) 1035-1051.

[Pfl 96] Pflug, G.Ch.: Optimization of stochastic models - the interface between simulation and optimization, Kluwer Academic Publishers, 1996.

[PJ 92] Polyak, B.T., Juditsky, A.B.: Acceleration of stochastic approximation by averaging, SIAM J. Control and Optimization 30 (1992) 838-855.

[Pra 91] Prasanna, V.K.(ed.): Special issue on massively parallel computation, J. Parallel and Distributed Computing 13 (1991) 123-.

[Pr 68] Prékopa, A.: Lineáris programozás I., Bolyai J. társulat kiadványa, 1968.

[Pr 70] Prékopa, A.: On probabilistic constrained programming, in: Proceedings of the Princeton Symposium on Mathematical Programming (Princeton University Press, Princeton, N.J., 1970) 113-138.

[Pr 71] Prékopa, A.: Logarithmic concave measures with applications to stochastic programming, Acta Scientiarium Mathematicarum (Szeged) 32 (1971) 301-316.

[Pr 73a] Prékopa, A.: On logarithmic concave measures and functions, Acta Scientiarium Mathematicarum (Szeged) 34 (1973) 335-343.

[Pr 73b] Prékopa, A.: Contributions to the theory of stochastic programming, Math. Programming 4 (1973) 202-221.

[PGDP 76] Prékopa, A., Ganczer, S., Deák, I., Patyi, K.: The STABIL stochastic programming model and its experimental application to the electrical energy sector of the Hungarian economy, in: Proc. of the International Symposium on Stochastic Programming, Oxford 1976, (ed. M.A.H. Dempster) Academic Press, 1980, 369-385.

[PS 78] Prékopa, A., Szántai, T.: A new multivariate gamma distribution and its fitting to empirical data, Water Resources Research 14 (1978) 19-24.

[Pr 88a] Prékopa, A.: Boole-Bonferroni inequalities and linear programming, Oper. Res. 36 (1988) 145-162.

[Pr 88b] Prékopa, A.: Numerical solution of probabilistic constrained programming problems, In: Numerical techniques for stochastic optimization (eds. Yu. Ermoliev, R. Wets), Springer series in computational mathematics, Springer Verlag, 1988, 123-139.

[Pr 95] Prékopa, A.: Stochastic Programming, in: Mathematics and its Applications 324, Kluwer, 1995.

[Rap 77] Rapcsák, T.: A SUMT módszer alkalmazása logaritmikusan konkáv feltételi függvényeket tartalmazó nem-lineáris programozási feladat megoldására, MTA Számítástechnikai és Automatizálási Kutató Intézet Közleményei, 19 (1978) 17-27.

[RM 51] Robbins, H., Monro, S.: A stochastic approximation method, Ann. Math. Stat. 22 (1951) 400-407.

[RW 87] Robinson, S.M., Wets, R.J.-B.: Stability in two-stage programming, SIAM J. on Control and Optimization 25 (1987) 1409-1416.

[RSch 87] Römisch, W., Schultz, R.: Stability of solution for stochastic programs with complete recourse, Mathematics of Operations Research, 18 (1993) 590-609.

[Rus 80] Ruszczyński, A.: Feasible direction methods for stochastic programming problems, Math. Programming 19 (1980) 220-229.

[RSw 97] Ruszczyński, A., Świętanowski, A.: Accelerating the regularized decomposition method for two-stage stochastic linear problems, European Journal of Operational Research 101 (1997) 328-342.

[RSh 03] Ruszczyński, A., Shapiro, A.(eds.): Stochastic Programming, Handbook of Stochastic Programming 10, 2003, Elsevier.

[Sch 83] Schittkowski, K. (1983): The numerical solution of constrained linear least-squares problems, IMA J. Numer. Anal. 3, 11-36.

[SDC 93] Sen, S., Doverspike, R.D., Cosares, S.: Network planning with random demand, Telecommun. Systems 3 (1994) 11-30.

[SS 90] Sen, S., Srivastava, M.: Regression analysis, Springer-Verlag, 1990.

[SH 98] Shapiro, A., Homem-de-Mello, T. : A simulation based approach to two-stage stochastic programming with recourse, Math. Programming 81 (1998) 301-325.

[Spa 92] Spall, J.C.: Multivariate stochastic approximation using a simultaneous perturbation gradient approximation, IEEE Trans. on Automatic Control 37 (1992) 332-341.

[Str 74] Strazicky, B.: On an algorithm for solution of the two-stage stochastic programming problem, Methods of Operation Research 19 (1974) 142-156.

[Sz 86] Szántai, T.: Evaluation of a special multivariate gamma distribution function, Math. Programming Study 27 (1986) 1-16.

[Sz 88] Szántai, T.: A computer code for solution of probabilistic constrained stochastic programming problems, in: Numerical techniques for stochastic optimization 1984, Springer series in computational mathematics (eds. Yu. Ermoliev, R. Wets.), Springer Verlag, 1988, 229-235.

[Sz 00] Szántai, T.: Improved bounds and simulation procedures on the value of the multivariate normal probability distribution function, Annals of Operations Research 100 (2000) 85-101.

[Tau 62] Tausworthe, R.C.: Random numbers generated by linear recurrence modulo two, Math. Computation 19 (1965) 201-209.

[Ton 90] Tong, Y.L.: The multivariate normal distribution, Springer Series in Statistics, 1990.

[Vaj 72] Vajda, S.: Probabilistic Programming, Academic Press, New York, 1972.

[VB 96] Vandenberghe, L., Boyd, S.: Semidefinite programming, SIAM Review March, 1996. 1-50.

[VL 02] Van Huffel, S., Lemmerling, P.: Total least squares and errors-in-variables modeling – analysis, algorithms and applications, Kluwer, 2002.

[Vij 97] Vijverberg, W.P.M.: Monte Carlo evaluation of multivariate normal probabilities, Journal of Econometrics 76 (1997) 281-307.

[Wal 77] Walker, A.J.: An efficient method for generating discrete random variables with general distributions, ACM TOMS 3 (1977) 253-256.

[WHS 96] Wallace, S.W., Higle, J., Sen, S.(eds.): Stochastic Programming: algorithms and models, Ann. of Operations Research, Baltzer, V. 64, 1996.

[Was 69] Wasan, M.T.: Stochastic approximation, University Press, Cambridge, 1969.

[Wet 83] Wets, R. J.-B.: Stochastic programming solution techniques and approximation schemes, in: Mathematical Programming: The State of the Art (eds. A.Bachem, M.Grotschel, and B.Korta), Springer, 1983, 566-603.

[Wet 88] Wets, R. J.-B.: Large scale linear programming techniques in stochastic programming, in: Numerical techniques for stochastic optimization 1984, Springer series in computational mathematics, (eds. Yu. Ermoliev, R. Wets), Springer Verlag, 1988, 65-94.

[WZ 99] Wets, R.J.-B., Ziemba, W.T.(eds.): Stochastic programming: State of the Art, Ann. of Operations Research, Baltzer, V.85, 1999.

[Wle 95] van der Wlerk, M.H.: Stochastic programming with integer recourse, Labyrint Publ. Capelle on de IJssel, The Netherlands, 1995.

[WSV 99] Wolkowicz, H.,Saigal, R., Vandenberghe, L.(eds.): Handbook of semidefinite programming, Kluwer Academic Publishers, 1999.

[Zan 69] Zangwill, W.I.: Nonlinear programming: a unified approach, Englewood Cliffs, N.J., Prentice Hall, 1969.

[Zen 89] Zenios, S.S.: Parallel numerical optimization: current status and an annotated bibliography, ORSA Journal on Computing (1989) 20-43.

[Yin 91] Yin, G.: On extensions of Polyak's averaging approach to stochastic approximation, Stochastics and Stochastics Reports 36 (1991) 245-264.